IMES DISCUSSION PAPER SERIES

耐タンパー性に基づくデジタル通貨ウォレットの研究動向
― 匿名性と透明性の両立に向けて ―

大塚（おおつか） 玲（あきら）

Discussion Paper No. 2022-J-9

IMES

INSTITUTE FOR MONETARY AND ECONOMIC STUDIES

BANK OF JAPAN

日本銀行金融研究所

〒103-8660 東京都中央区日本橋本石町 2-1-1

IMES Discussion Paper Series 2022-J-9
2022 年 10 月

耐タンパー性に基づくデジタル通貨ウォレットの研究動向
― 匿名性と透明性の両立に向けて ―

大塚（おおつか） 玲（あきら）*

要　　旨

近年、デジタル通貨に関する研究開発が活発化している。デジタル通貨には、マネーロンダリング及びテロ資金供与への対応として、秘匿されたデータを開示できる性質が求められることから、匿名での取引を可能とする性質（匿名性）をもたせたうえで、必要に応じて取引の内容を開示できる性質（透明性）を確保できることが望ましい。研究コミュニティでは、1980 年代から匿名性と透明性を両立するトークン型デジタル通貨が研究されてきており、ブロックチェーンを基盤とするデジタル通貨への応用も研究され始めている。しかし、アンチマネーロンダリング等の社会的要求を満たす透明性の技術特性は明らかでなく、暗号技術だけで社会的要求に応える匿名性と透明性の両立は難しい。そこで、本稿では、透明性を柔軟に実現できる可能性のある理論基盤として、最近提案された実行証跡付セキュアプロセッサ(Attested Execution Secure Processor, Pass, Shi and Tramér, 2017) に基づく方式の実現可能性を考察する。近年のスマートフォンに搭載された耐タンパー技術は、この理論が求める理想的なプラットフォームの要件をほぼ満たしており、匿名性と透明性を両立可能であると考えられる。

キーワード: 暗号資産、証跡実行セキュアプロセッサ、耐タンパー技術、
デジタル通貨ウォレット、透明性、匿名性
JEL classification: E49、L86、L96、Z00

* 情報セキュリティ大学院大学教授（E-mail: otsuka@iisec.ac.jp）

本稿は、筆者が日本銀行金融研究所客員研究員の期間に行った研究をまとめたものである。本稿の作成に当たっては、櫻井幸一教授(九州大学) から有益なコメントを頂戴した。また、金融研究所スタッフに貴重なコメントを頂戴した。ここに記して感謝したい。ただし、本稿に示されている意見は、筆者個人に属し、日本銀行の公式見解を示すものではない。また、ありうべき誤りはすべて筆者個人に属する。本稿は、2022 年 4 月 15 日時点の情報を元に執筆された。

目 次

1　はじめに

未来のデジタル通貨は、利用者が自身の個人情報を自らコントロールできる性質を持つことが望ましい。一方で、現金が持つ高い匿名性を実現すべきか否かは議論が分かれている。匿名性のないデジタル通貨が普及すれば、発行体や関係者に悪意がない場合でも、規制当局や取引当事者による利用者のデータの保護の失敗等によるセキュリティリスクを排除できない。他方、デジタル通貨は高額の資金を瞬時に移動できることから、少なくとも金融機関に求めていたものと同程度あるいはそれ以上のアンチマネーロンダリング（AML）やテロ資金供与への対策（CFT）が求められる。そこで、本稿では、デジタル通貨ウォレット機能を担う多様なオープンソースプログラムの開発を推奨し、規制当局が AML/CFT 対応ルールを満たすオープンソースプログラムを選別したホワイトリストを公開することで、利用者がこのホワイトリストの中から信頼できるものを選択して自身の耐タンパーウォレットにインストールして実行するシナリオを想定する。耐タンパーウォレット内で動くプログラムをオープンソース化することで、当局による過度なプライバシー侵害等への不信感を払拭しつつ、社会的に合意の取れた AML/CFT 対応ルールの技術的強制による遵守を同時に達成できるシステム（Regulatory Technology）の実現可能性を考察する。この際、デジタル通貨による資金決済において AML/CFT ルールが適切なプログラムによって正しく履行されたことを、取引当事者や規制当局等が検証可能とするための枠組みとして、近年提案された実行証跡付セキュアプロセッサ (Pass, Shi and Tramér, 2017) の応用を提案する。

デジタル通貨に関する研究では、これまでも匿名での取引を可能とする性質（匿名性）と必要に応じて取引の内容を開示させることができる性質（透明性）の両立が議論されてきた。特に、(1) 支払者は匿名で取引が可能であるが、受領者についてはデジタル通貨の換金にあたり金融機関に対して匿名性をもたないもの (Chaum, 1982; Chaum, Grothoff and Moser, 2021)、(2) 通常の取引は全て匿名だが、裁判所等の令状に基づき、特定の人物に関わる取引や特定の取引を指定して当事者を開示できる機能を備えたもの (Stadler, Piveteau and Camenisch, 1995) 等が提案されている[1]。これらは、暗号理論で実現できる信頼性の高い方式で、AML や CFT にも一定の効果が期待される。

匿名性と透明性の両立に関して、本稿ではより現実的な選択肢として、耐タンパー技術を用いたデジタル通貨ウォレット技術に注目する。耐タンパー技術は、半導体デバイスの回路構造のランダム化やチップ損傷の検知機能に関する技術であり、暗号鍵等の機密情報を半導体チップ（セキュアエレメント）の内部に格納することで、機密情報を保護しながら、デジタル署名等の暗号処理を行う技術である。IC カードや SIM カード等で古くから利用されているが、当時はあらかじめ固定化されたプログラムを半導体デバイス内に埋め込む必要があった。近年では、事後的にプログラムの更新や新規導入が可能な設計となっており、こうした半導体デバイスとして、eSIM[2]や FeliCa チップ[3]がスマートフォンに搭載されるようになっている。プログラム可能な耐タンパーデバイスは、GlobalPlatform[4]が定めた仕様に従っているものが多く、それらは GP-SE[5]と総称されている。マイナンバーカードのスマートフォン搭

[1]他方、ブロックチェーンで実現されている匿名性 (Ben-Sasson et al., 2014; Maxwell, 2015; Heilman et al., 2017; Bünz et al., 2020) は、透明性を考慮していないものが多く (大塚, 2022) 、透明性の実現を目指した研究 (Mitani and Otsuka, 2020) は少ない。

[2]eSIM: embedded Subscriber Identity Module

[3]FeliCa チップ: 交通系 IC カードやおサイフケータイとして利用されている耐タンパーデバイス。

[4]セキュアエレメント技術の標準化を推進する非営利民間企業。

[5]GP-SE: GlobalPlatform 仕様に準拠したセキュアエレメント。

載プロジェクト[6]でも、GP-SE にマイナンバーカードの機能を搭載し、スマートフォンだけで確実なリモート個人認証（eKYC）を実現するためのインフラ作りが進められている。デジタル通貨においても、スマートフォンに搭載された GP-SE 等の耐タンパー技術を利用することにより、スマートフォンならではの利便性や受容性を背景にして、匿名性と透明性を両立するデジタル通貨の実現が期待される。

以下、本稿では、2 節において、スマートフォンの GP-SE でデジタル通貨ウォレットを実現する際の安全性を考察する。続く 3 節では、耐タンパーデバイスを用いて、透明性の高い取引の実行を利用者に強制するオブザーバー（Observer）方式 (Brands, 1993) を紹介する。4 節では、任意のプログラムを搭載可能な耐タンパーデバイスの暗号学的なモデルである実行証跡付セキュアプロセッサ (Pass, Shi and Tramér, 2017)[7]を紹介し、デジタル通貨の匿名性と透明性を両立する技術の実現可能性を考察する。

2　デジタル通貨ウォレットの安全性

デジタル通貨のスキームは、デジタル通貨を発行する主体とその利用者で構成される。デジタル通貨の利用者は、デジタル通貨ウォレット（以下、ウォレット）と呼ばれるデバイスを利用してデジタル通貨の保管や移動を行う。そのため、ウォレットには、(1) デジタル通貨の保管や移動に必要となるデータの保護、(2) 取引実行時の本人意思の確認が求められる。以下では、暗号資産での利用が普及しているコールドウォレット[8]を例に、(1) と (2) の機能がどのように実現されているかを概観する。

(1)　コールドウォレット

イ　デジタル通貨の保管・移動に必要なデータの保護

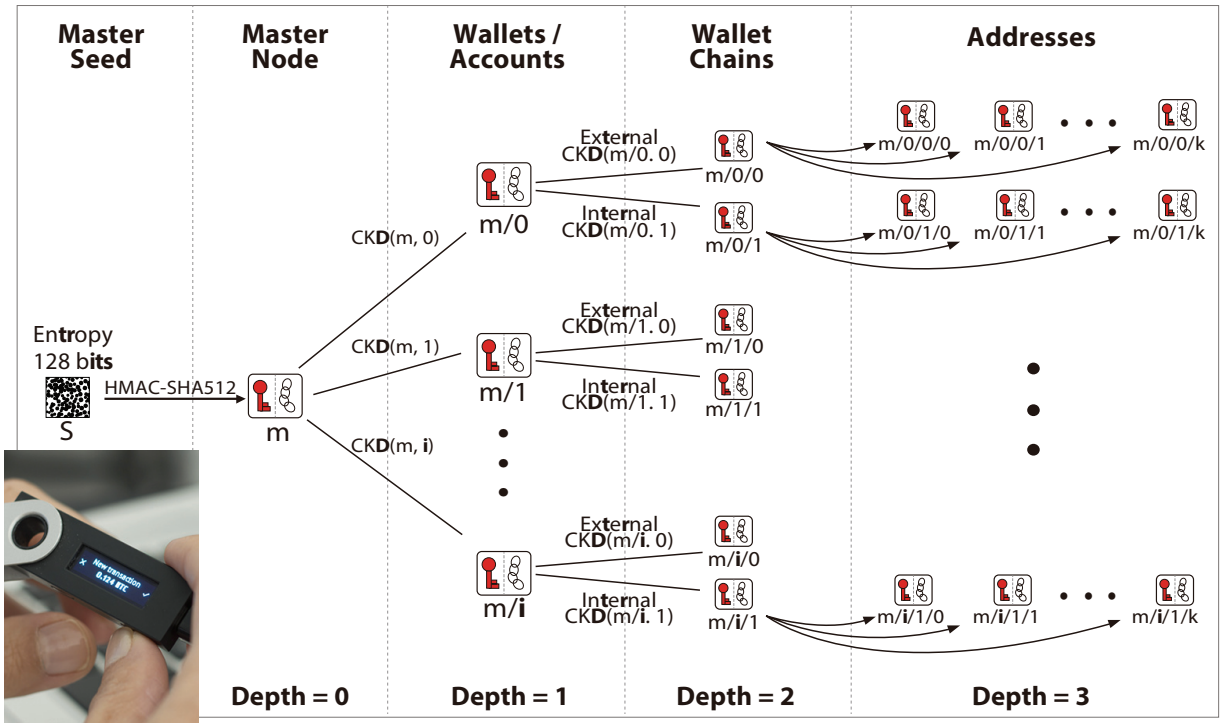
ブロックチェーンをベースとする暗号資産取引では、暗号資産の保管や移動を暗号処理によって行っている。すなわち、ウォレットには秘密鍵が保管されており、それを用いて生成したデジタル署名を取引内容に付して送付すると、ブロックチェーンネットワークによって暗号資産の移動が行われる。このように、ウォレット内に暗号資産そのものが保管されているわけではないが、秘密鍵によって暗号資産を移動できるという点で、ウォレット内での秘密鍵の管理が暗号資産の管理に直結しているといえる。一般的なコールドウォレットは耐タンパーデバイスで実現されており、秘密鍵は耐タンパーデバイスの内部で生成され、正規以外の方法では外部に取り出すことができない構成になっている。

また、Bitcoin や Ethereum では、ブロックチェーン上で複数のアカウントを扱うことが可能であり、その際には対応する秘密鍵を複数管理する必要がある。また、暗号資産の流れを秘匿することを目的に、アカウントを使い捨てにすることも可能であり、そうした場合にも秘密鍵を多く管理する機構が求められる。こうした機構を実現するウォレットとして、1

[6]マイナンバーカードの機能のスマートフォン搭載等に関する検討会（総務省）、https://www.soumu.go.jp/main_sosiki/kenkyu/mynumber_smartphone/index.html

[7]Attested Execution Secure Processor(Pass, Shi and Tramér, 2017) の訳語に、原語のままとする案や実行証明機能付きセキュアプロセッサなどの訳語も検討したが、本論文では現在の実行証跡付セキュアプロセッサとした。

[8]暗号資産の分野では、インターネットから切り離して保管できるデバイスをコールドウォレット、インターネットに常時接続して利用するタイプをホットウォレットと呼んでいる。コールドウォレットは、暗号資産の移動時にのみシステムに物理的に接続するものであるため、不正アクセスのリスクが相対的に低い。



(a) コールドウォレットの一例（出典：bitcoin.org）

(b) HD ウォレット（BIP-32）の仕組み（出典：`https://github.com/bitcoin/bips/blob/master/bip-0032.mediawiki`）

図 1: 暗号資産における鍵管理技術

つのマスター乱数シード (S) から全ての鍵を導出する階層型決定性ウォレット（HD-Wallet: Hierarchical Deterministic Wallet[9]）が普及している。決定性とは、同じマスター乱数シードを用いる限り、常に同じ鍵を導出できる性質である。階層型決定性ウォレットであれば、紛失または故障によりウォレットが使用不能になった際でも、使用開始時に出力されるマスター乱数シード S を保管しておくことで[10]、いつでも別のウォレットに鍵を再構成することができる。また、鍵は図 1（b）に示すようなマスター乱数シード S を根とするツリー構造を持ち、アドレスで使用する鍵は、マスター乱数シード S から一意に特定できるようになっている。

ロ　取引実行時の本人意思の確認（WYSIWYS）

デジタル通貨のスキームでは、サーバと利用者間で取引内容の通信が行われる。取引実行時に利用者の意思を確認するにあたっては、サーバから送られた通信内容が利用者に正確に表示され、かつ、利用者の入力内容が改ざんされずにサーバに伝達されるといった、人間とシステムとの間の安全な通信路の構築が極めて重要となる。これはトラステッド・パス

[9]HD-Wallet の基本仕様は BIP-32(Bitcoin Improvement Proposals）で定義される。BIP とは、ビットコイン開発者らによって作成されたビットコインシステムを改善するための提案のことであり、議論や投票を経てビットコインコミュニティにおける標準規格となっているものもある。

[10]耐タンパーデバイス内で生成された S は、使用開始時に限って取り出すことが可能であり、別の耐タンパーデバイスに複製保管しておくことができる。ただし、安全な実装では、使用開始後に S を取り出すことはできない。

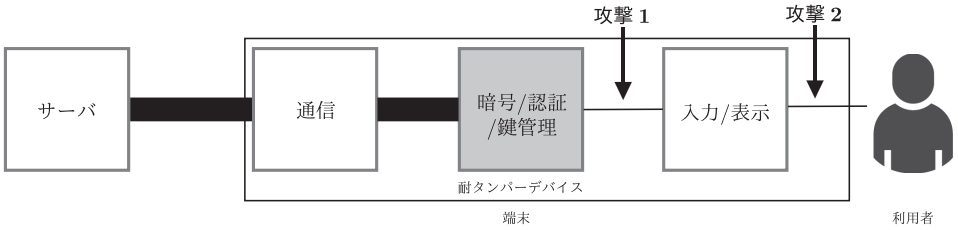


図 2: 利用者-サーバ間の通信路と攻撃箇所

(National Computer Security Center, U.S., 1985) として、1980 年代から提唱されている概念であるが、多くのコンピュータシステムにおいて実現が困難であるのが実情である。例えば、フィッシングやインターネットバンキングにおける不正アクセスは、トラステッド・パスの構築が困難であるために起こりうるものである。

デジタル通貨を発行する主体のサーバ（以下、サーバ）と利用者の端末（耐タンパーデバイス）からなるシステムを例に挙げ、サーバが端末に送信したメッセージを利用者が承認する手順を考える（図 2)。なお、耐タンパーデバイスは、暗号、認証、鍵管理の 3 つの機能を有しているものとする。サーバがメッセージを暗号化して端末に送信すると、端末はメッセージを復号してディスプレイに表示する。利用者は表示内容を確認し、キーボード等から暗証番号を打ち込むことにより承認を与え、メッセージに対する利用者の署名が生成される。このとき、サーバから秘密鍵を管理している耐タンパーデバイスまでの経路は暗号技術等で保護することができる。しかし、システムがマルウェアに感染等していた場合、メッセージが耐タンパーデバイスで復号されて平文になったところで表示を変更される（攻撃 1）リスクがあるほか、利用者の入力内容が改ざんされる（攻撃 2）リスクもある。このように、耐タンパーデバイスを用いた場合であっても、マルウェアへの感染等により利用者が意図しないメッセージに対して署名してしまうリスクがある。攻撃 1、攻撃 2 に対抗する技術的な対策により、利用者が確認している表示内容と、耐タンパーデバイスの署名内容を一致させる性質を、WYSIWYS（What You See Is What You Sign）と呼ぶ。容易に見えて、実は達成が困難な性質である。

この問題に対処するために、Weigold らは、表示機能に加え「OK」と「キャンセル」の 2 つの入力ボタンを備えた USB デバイスを試作した (Weigold et al., 2008)。本 USB デバイスは、PC に接続してサーバとの間で TLS（Transport Layer Security）認証を行うことで、サーバと USB デバイスの間に暗号通信による安全なチャネルを確立し、経路上の PC に存在するマルウェア等に妨害されることなく、サーバから送信された情報をそのまま USB デバイス上で表示／確認することで確実に WYSIWYS を達成するものである。これは、図 2 の枠内にある通信、暗号／認証／鍵管理、入力／表示の機能を一体化した専用の耐タンパーデバイスで構成することにより、攻撃 1、攻撃 2 に対抗してサーバのメッセージを確実に利用者に伝達する解決策になっている。

コールドウォレットにも同様の機能を備えているものが多い。図 1（a）に示すように、一般的なコールドウォレットにも、液晶表示とボタンが装備されている。これは、先行研究と同じく、通信、暗号／認証／鍵管理、入力／表示の全ての機能を一体のデバイスとして実現するアプローチに従っている。このアプローチは安全性を高める最も確実な方法だが、専用のデバイスを準備しなければならないことからコストや利便性の面で課題が残る。

(2)　スマートフォンにおけるデジタル通貨ウォレット

イ　実現が見込まれるセキュリティ機能

スマートフォンには複数のセキュリティハードウェアが備わっている。代表的なものには、TEE[11]や、前述の FeliCa チップがある。特に、TEE は多くの ARM プロセッサに搭載され、一般のアプリケーションから隔離された処理が可能なことから、生体認証や暗号処理などの OS 標準のセキュリティ機能の実装に用いられている。一方、FeliCa チップは、CC 認証（ISO/IEC 15408）[12]における AVA_VAN.5[13]等の網羅的なペネトレーションテスト（攻撃耐性の評価）を通過したチップである。以前は、電子マネー・交通系 IC 等の専用チップであったが、2019 年からは GP-SE としての機能が導入され、多様なアプリケーションに利用できるようになった。2019 年冬から Android スマートフォンへの搭載が開始され、2021 年末時点では国内で新たに発売されたほぼ全ての Android スマートフォンに GP-SE が搭載されている。

加えて、マイナンバーカードのスマートフォン搭載での検討状況[14]を考慮すると、スマートフォン内の GP-SE にウォレットを搭載することが可能であり、その場合には以下の機能も組み込める可能性がある。

1. 生体認証による本人確認

   Android 端末の生体認証機能は、前述の TEE を用いた暗号鍵管理機能（CryptoObject）と連動することで、生体認証が受理されたことを暗号鍵による認証プロトコルで証明することができる。これは、生体認証が受理されない限り CryptoObject 内の鍵を利用できないように構成するものであり、暗号鍵を用いた認証プロトコルが成功すれば、端末に正しい生体が提示されたことを外部から推定できる。こうした機能を用いた生体認証の規格には FIDO[15]があり、リモートからの本人認証に広く利用されている。また、マイナンバーカードのスマートフォン搭載に関する検討では、この機能を用いて、パスワードによる認証を生体認証に置き換え可能とすることが計画されている[16]。同様に、デジタル通貨ウォレットのロック解除にも、フィッシングによる漏洩リスクの高いパスワードではなく、生体認証による本人確認を安全に用いることが考えられる。

2. タッチ動作による支払い

   スマートフォンの多くには NFC[17]が備わっており、対面販売や自動販売機に対してもタッチ動作によるデジタル通貨ウォレットからの支払いが技術的に可能である。マイナンバーカードのスマートフォン搭載に関する検討では、NFC 経由で受信されたメッ

[11]TEE: Trusted Execution Environment。安全な処理を実行するための隔離実行環境を備えたプロセッサ。ARM の TrustZone、Intel の SGX（Software Guard Extensions）等。

[12]CC（Common Criteria）は国際標準規格 ISO/IEC 15408 で規定された IT システムのセキュリティ保証レベル定める規格であり、各国で CC を用いたセキュリティ認証制度が運用されている。

[13]AVA_VAN.5 とは、AVA（脆弱性評定）クラスの VAN 脆弱性分析ファミリである。5 は最高レベルの脆弱性分析を示す。例えば、以下の文献を参照。IPA：スマートカードへの攻撃能力の適用（ver 2.9）、https://www.ipa.go.jp/security/jisec/hardware/documents/CCDB-2013-05-002_J.pdf、2013 年 5 月。

[14]総務省: 第 1 次とりまとめ ～電子証明書のスマートフォン搭載の実現に向けて～ 2020 年 12 月 15 日。

[15]FIDO（Fast IDentity Online）は、FIDO アライアンスが定めるリモート認証の規格。

[16]総務省: マイナンバーカードの機能のスマートフォン搭載等に関する検討会（第 5 回）、2021 年 3 月 3 日。

[17]NFC: Near Field Communication。スマートフォンのタッチ動作による無線通信のインターフェイス。

セージを GP-SE 内の JPKI アプレット[18]にルーティング（経路指定）することで、住民票のコンビニ交付や健康保険証等による資格確認等への応用が検討されている[19]。

3. 紛失／盗難や機種変更時の対応

スマートフォン紛失・盗難時の対応として、スマートフォンの通信機能を使い、遠隔から GP-SE 内の情報を消去することが技術的に可能である[19]。これは従来の IC カードにはなかった機能であり、利用者に一定の安心感を与えられる機能であると思われる[20]。マイナンバーカードのスマートフォン搭載に関する検討では、機種変更の際、新しいスマートフォンの GP-SE で新たに鍵情報を生成すると共に、古いスマートフォン内の鍵情報を遠隔消去するよう検討が進められている[19]。機種変更時には、古いスマートフォンの所有者であることを確認するのではなく、マイナンバーカードによって本人確認を行う方向で検討が進められており、スマートフォン紛失・盗難への対策が講じられる予定である。デジタル通貨のケースにおいても、古いスマートフォンの GP-SE 内の鍵情報を消去する機能は必要となるが、加えて、新旧機種間でのデジタル通貨（を管理する暗号鍵）の移動が確実に行われることの保証が必要なため、機種変更の手順はより複雑になると想定される。例えば、前述の階層型決定性ウォレットを採用し、マイナンバーカードの鍵を用いてマスター乱数シードを暗号化してクラウド等で管理しておくことで、クラウドから新しいスマートフォンにマスター乱数シードを復元するなどの手順が考えられる。

GP-SE によって上記セキュリティ機能を達成できると見込まれるが、GP-SE はスマートフォンに常時接続されていることから、ウォレットにはマルウェア等からの不正な署名要求が寄せられるリスクがある。そのため、必要な時だけ接続するコールドウォレットと比較してリスクが高く、不要な署名要求を識別して拒絶する機能が不可欠である点には留意が必要である。以下、GP-SE のリスクについて考察する。

ロ　取引実行時の本人意思の確認（WYSIWYS）

スマートフォンに搭載するウォレットは、GP-SE 内にアプレットの形で実装され、取引内容の入力・表示は Android App に分離した形で実装される。WYSIWYS を達成するためには、これらが一体として動作し、正しい取引内容を表示すると共に、本人意思に沿った入力を GP-SE に正しく伝達する機構が必要となる。以下では、GlobalPlatform 規格に定義されているアクセス制御機構を用いて WYSIWYS を達成する方式を説明する。この機構は多くのスマートフォン OS（Android OS、iOS 等）に既に標準実装されており、スマートフォン上の特定のアプリと GP-SE を一体化して動作させることができるようになっている。

GlobalPlatform 規格におけるアクセス制御機構は、図 3 に示す構造になっている。ここで、セキュアエレメント（Secure Element、GP-SE）は物理的にはデバイス内に埋め込まれているが、機能的にはデバイスと分離されているため図では外側に描かれている。スマート

[18]マイナンバーカードの JPKI 機能を実現する Java アプレットで GP-SE 内に格納されている。

[19]総務省: マイナンバーカードの機能のスマートフォン搭載等に関する検討会（第 2 次とりまとめ）、2022 年 4 月 15 日。

[20]ただし、通信が常時遮断された環境では機能せず、GP-SE 内の情報の利用休止や消去には確実な本人確認が求められることから利用停止までの時間差が生じ、紛失／盗難リスクを完全に回避できる技術ではないことには注意が必要である。

フォン上のアプリである Device Application にはコード署名（Signature）[21]が付与されており、それを Access Control Enforcer[22]が認識し、セキュアエレメントへのアクセス制御を実行する。セキュアエレメントの内部は、製造者が事前に設定した Issuer SD（発行者セキュリティドメイン）と、セキュアエレメントの利用目的に応じて導入可能な Application Provider SD（サービス提供者セキュリティドメイン）に分かれている。それぞれの SD は異なる鍵で記憶空間が暗号化され、SD を越えて機密情報が漏洩しないように隔離された空間となっている。各 Application Provider SD は、固有のアクセス制御ルールが格納されたデータベース ARA-C（Access Rule Application Client）を保持し、アクセス制御ルールを満たすアクセス要求だけが Application Provider SD 内でデジタル署名や決済処理などの機密情報を扱う SE Application に接続される。各 ARA-C のルールは、Issuer SD 内の ARA-M（Access Rule Application Master）に統合され、アクセス制御ルール全体の無矛盾性が検証された後にデバイス内の Access Control Enforcer と共有される。

次にアクセス制御機構の動作をより詳しく述べる。アクセスを許可されるアプリケーションは、事前に ARA-C のアクセス制御データ[23]にコード署名を検証するための公開鍵を登録しておく。Access Control Enforcer は、この公開鍵で Device Application のコード署名を検証できる場合に限り、対応する Application Provider SD 内の SE Application へのアクセスを許可する (GSM Association, 2011)。こうした機構により、リパッケージ攻撃[24]等で紛らわしいフィッシングアプリが起動されたとしても、ARA-C にコード署名検証用の公開鍵が登録されていない同アプリは SE Application にアクセスできないことが保証される[25]。

以上から、攻撃 1（メッセージが耐タンパーデバイスで復号されて平文になったところで表示を変更されるリスク）に関しては、暗号／認証／鍵管理を担う GP-SE とスマートフォンアプリをアクセス制御機能により一体化することにより、対策を講じることができる。

また、攻撃 2（利用者の入力内容が改ざんされるリスク）に関しては、改ざんした表示を重ねて錯誤させるスクリーンオーバレイ攻撃[26]が主要な攻撃として知られている。ただし、こうしたリスクは比較的早くから認識され、OS による対策も進んでいる。例えば、Android6.0 以降の OS ではパーミッションを取得したアプリケーションでなければスクリーンオーバレイを実行できない仕様に改善されている。

3　耐タンパー性に基づく匿名性・透明性両立の先行研究

本節では、GP-SE を基盤としたウォレットの構成を前提として、匿名性と透明性を兼ね備えたデジタル通貨の実現方法を考察する。

オブザーバー方式とは、利用者による不正を防止するために、耐タンパーデバイスを利用者

[21]ソフトウェアに対して製造元が付したデジタル署名。利用者はデジタル署名の検証により、ソフトウェアの真正性と流通経路で改ざんされていないことを確認することができる。

[22]セキュアエレメントに格納されているアクセスルールを取得し、それらのルールを適用して制限するソフトウェア。

[23]アクセス制御データ（Access Control Data）は、アクセスを許可するコード署名検証用の公開鍵データ。

[24]既存の正規アプリに悪意のあるコードを組み込み再配布する行為。

[25]アクセス制御機構（図 3 中 Access Control Enforcer) は Android OS の一部として実装されている。このため、端末を UNLOCKED 状態にし、Root 化した状態の Android OS を改ざんすればアクセス制御機構を無効化することも可能ではあるが、通常、利用者が意図的に実施しない限り、この操作は困難となるように設計されている端末が多い。

[26]実行中の他のアプリやウィンドウに別のウィンドウを重ねて表示する攻撃であり、本来のウィンドウをクリックしているようにみせかけて実際には別のウィンドウをクリックさせるもの。

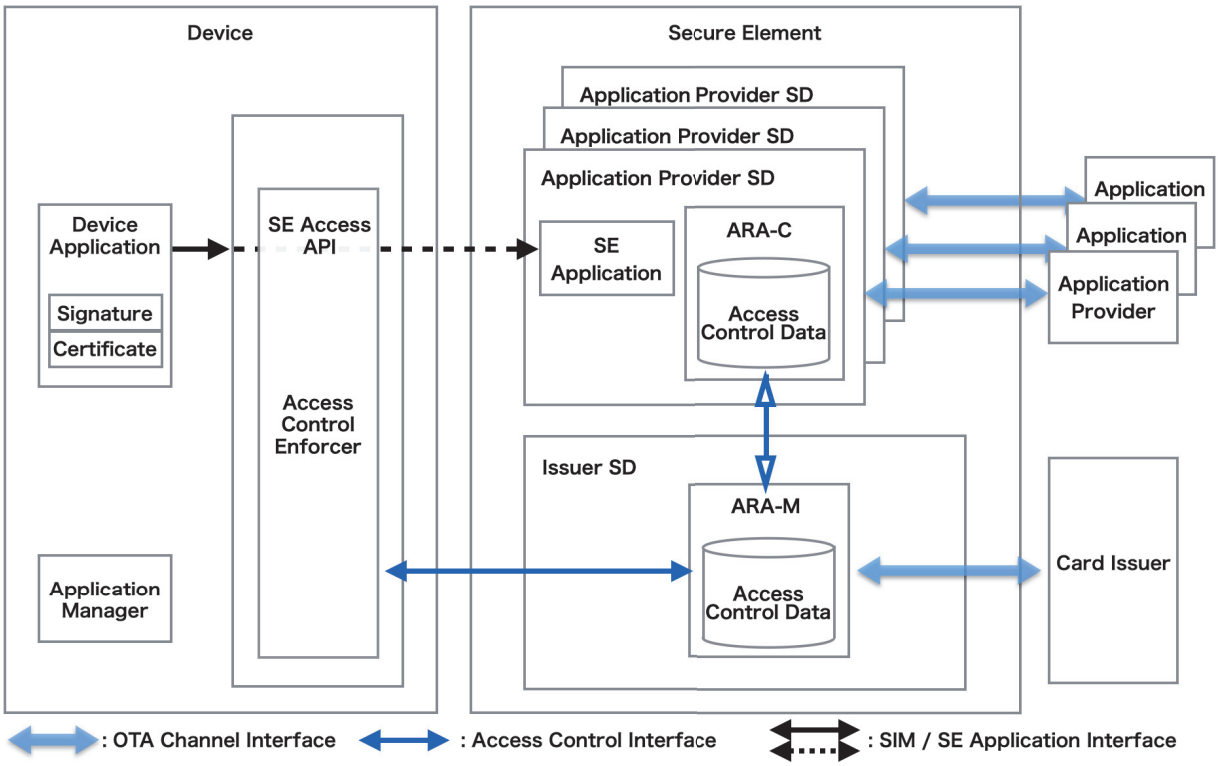


図 3: スマートフォン（Device）上のアプリ（Device Application）から GP-SE（Secure Element）へのアクセス制御機構 (出典: GlobalPlatform, 2014)

のオブザーバー（監視者）として利用するものである。オブザーバーを利用者の端末に搭載することで、透明性の高い取引の実行を利用者に強制することができる (Chaum and Pedersen, 1993; Brands, 1993)。具体的には、トークンの二重支払いが発生した際にのみ事後的に利用者の特定を可能にすることで、匿名性[27]と透明性をデジタル通貨のスキームに持たせるものである。本節では、オブザーバー方式によってデジタル通貨の使用を一回のみに制限する義務（二重支払いの防止）を技術的に強制する例を示す[28]。

なお、本稿では銀行がデジタル通貨の発行や償還を担うことを想定している。銀行は本人確認済みの預金者（利用者）に対して、預金口座の残高を減じることで見合いのデジタル通貨をトークンとして発行する。また、預金者（受け手／店舗）からトークンを受け入れたら当該預金者の預金口座に相当額を加算し、当該トークンを償還・償却することを想定している。これは、デジタル通貨の発行・利用・償還においてクローズドループ型（図 4）を想

[27] Chaum や Brands ら (Chaum and Pedersen, 1993; Brands, 1993) のオブザーバー方式における匿名性とは、支払者の匿名性のみを実現するものであり、トークン償還銀行に対する受領者の匿名性はない。90 年代当時の耐タンパーデバイスは、出荷前に耐タンパーデバイス内のプログラムを固定し、出荷後は変更を加えさせないことで安全性を高めるという考え方が主流であったため、当時は、シンプルで完結した方式を提示しなければならなかった。一方、後述する実行証跡付セキュアプロセッサ方式では、前述の GP-SE に代表されるような、出荷後に耐タンパーデバイス内のプログラムを変更しても安全性を保つことができる機構を前提とする、2000 年代以降に確立した考え方に基づいているため、プログラムにより受け取り側の匿名性や顕名性を自在にコントロールすることができる。

[28] この考え方は、裁判所の令状がある場合のみ匿名取引の取引内容を開示する義務を強制するエスクローキャッシュ等 (Stadler, Piveteau and Camenisch, 1995; Fujisaki and Okamoto, 1998; Abe and Ohkubo, 2001) への応用も考えられる。

定していることを意味する[29]。

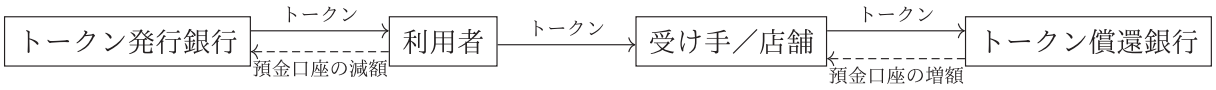


図 4: 銀行預金とクローズドループ型トークンの構成

(1) トークン型デジタル通貨における二重支払いへの対策

トークン型デジタル通貨とは、物理的な現金の代わりに暗号処理を施した「トークン」と呼ばれるデータを受け渡すことによって金銭価値を移転させる方式である。トークン型では、トークンの複製による二重払い（複製したトークンを支払いに使用すること）への対策が重要となる。

二重支払いへの対策には、これまで大きく 2 つの方式が研究されている。1 つ目はトークン償還銀行が支払い済みトークンのリストを保有し、店舗がトークンを受け取る際にトークン償還銀行に問い合わせる方式である。これには、全ての取引に対してオンラインで迅速に回答することが求められるため、トークン償還銀行に高速大容量なサーバ設備の設置が求められる。2 つ目は店舗がトークンを受け取る際にトークン償還銀行に照会せずに二重支払いを抑止もしくは防止しようとするものである。抑止する方法としては、トークン生成時に支払者の ID を暗号化して埋め込むことで、二重支払いされた 2 つのトークンから支払者 ID を特定できるようにするアプローチ (Chaum et al., 1990; Okamoto and Ohta, 1992; Brands, 1993) がある[30]。また、防止する手法としては、耐タンパーデバイスでトークンの使用を 1 回に制限するすることで二重払いを防止するアプローチがある。

ここで紹介するオブザーバー方式は、トークン償還銀行に照会することなく二重支払いを防ぐ上記 2 つの方法を融合した方式である。オブザーバーが耐タンパー性を有する場合には、トークンの使用を 1 回に制限することで二重支払いのリスクを排除できるほか、万が一、耐タンパー性が無効化された場合であっても、二重支払いが検出された場合には支払者の ID を事後的に特定することができる。

以下では、まず、トークンの偽造耐性に用いられる Schnorr 署名を説明する[31]。デジタル署名は秘密鍵を用いて生成され、秘密鍵と対をなす公開鍵を用いることで、デジタル署名の生成者を特定すると共にメッセージの改ざんの有無を検証することができる技術である。トークン偽造耐性とは、トークン発行銀行以外によるトークンの生成および改ざんが困難であることを指し、シリアル番号にトークン発行銀行のデジタル署名を施すことで構成することができる。(3) では、デジタル通貨に匿名性を持たせる方式として提案された Schnorr ブラインド署名を紹介する。Schnorr ブラインド署名は、ユーザがランダムに選んだシリアル番号を知らせることなく、トークン発行銀行にデジタル署名を付与させることができる方式である。トークン発行銀行による署名とトークンとの対応関係を秘匿できることから、トークン発行銀行とトークン償還銀行が結託して追跡を試みても、支払に利用されたトークンか

[29] クローズドループ型は、トークンの流れが一方向であり、利用者と店舗の役割が異なるタイプをいう。利用者が店舗に支払ったトークンは銀行に還流され、トークンに見合いの現金が店舗の銀行口座に振り込まれる。これに対し、オープンループ型は、店舗や利用者間の転々流通を可能とするタイプを指す。

[30] この分野の包括的なサーベイは、(中山他, 1997; 中山, 1998; 中山・太田・松本, 1999) に詳しい。

[31] デジタル署名には RSA や ECDSA など様々な方式があるが、ここではオブザーバーの考え方を具体的に解説するために、(Brands, 1993) がベースとしている Schnorr 署名を紹介する。

表 1: 本稿で紹介するデジタル通貨技術の比較

	トークンの偽造耐性	匿名性	透明性		
			二重支払いの事後検知	支払ルールの強制	多様なルールの強制
Schnorr 署名	○	なし	なし	なし	なし
Schnorr ブラインド署名	○	○	なし	なし	なし
追跡可能 Schnorr ブラインド署名	○	○	○	なし	なし
オブザーバー	○	○	○	○	なし
実行証跡付セキュアプロセッサ	○	○	○	○	○

らその利用者を特定することが困難となっている。さらに、(4) において、トークンの二重支払いを行った利用者を特定可能とする追跡可能 Schnorr ブラインド署名を紹介する。そのうえで、(5) では、オブザーバー方式を紹介し、次節で実行証跡付セキュアプロセッサを利用したデジタル通貨について考察を行う。なお、以下で紹介する方式の比較は表 1 を参照されたい。

(2) Schnorr 署名

トークン型のデジタル通貨は、乱数をトークンのシリアル番号として用い、シリアル番号に対してトークン発行銀行のデジタル署名を施すことで実現される。

本節では、オブザーバー方式の基本となる Schnorr 署名を説明する。Schnorr 署名は、乗法群上の離散対数問題の困難性に安全性の根拠をもつ方式である。離散対数問題とは、g と h が与えられたときに、$x = \log_g h$ となる x を求める問題である。つまり、Schnorr 署名は、公開鍵 h に対して、離散対数の解となる秘密鍵 x $(= \log_g h)$ を知っていることの証明であり、x を知らないものによる偽造が困難である。(Okamoto, 1992; Pointcheval and Stern, 2000; Seurin, 2012; Baldimtsi and Lysyanskaya, 2013)。

なお、システムパラメータとして、素数位数 q の乗法群 $\mathbb{G}$ と生成元 $g \in \mathbb{G}$ が定められ、公開鍵の生成元 g が公開されているほか、署名者は予め秘密鍵-公開鍵ペア $(x, h = g^x)$ を計算し、公開鍵 h を検証者に通知しているものとする[32]。

【署名生成】

- 署名者は、秘密鍵-公開鍵ペア $(x, h = g^x)$ を計算し、公開鍵 h を全利用者に通知する。
- 署名者は、乱数 $w \in \mathbb{Z}_q^*$ を用いて、$a = g^w$ を計算する。
- 署名者は、秘密鍵 x を用いて、メッセージ m と a のハッシュ値 $c = \mathcal{H}(m, a)$ に対し、$r = cx + w$ を計算する。
- 署名者は、(a, r) をデジタル署名として発行する。

[32]本節の演算は基本的に乗法群 $\mathbb{G}$ 上で行われるものとする。ただし、加法を含む演算は法 q の有限体 $\mathbb{Z}_q$ 上で行う。

【署名検証】

- 検証者は、検証者はメッセージ m とデジタル署名 (a, r) を受信し、ハッシュ値 $c = \mathcal{H}(m, a)$ を計算する。
- 検証者は、署名者の公開鍵 h に対して、$g^r = h^c a$ が成立することを検証し、秘密鍵 x を知りうる署名者によるデジタル署名であることを確認する[33]。

(3) Schnorr ブラインド署名

イ ブラインド署名によるデジタル通貨トークンの発行・使用

ブラインド署名とは、署名対象となるメッセージを署名者にみせることなく署名してもらう手法である。ブラインド署名を利用したデジタル通貨では、利用者が生成したシリアル番号に対する銀行のブラインド署名をトークンとして利用する。発行体である銀行がトークンの元となる情報を発行するが、これを用いた計算により利用者がトークンを構成するため、トークン情報と元情報との対応関係はトークン発行銀行とトークン償還銀行に対して秘匿される。それゆえ、トークンが匿名性を持つことになる。すなわち、トークンを構成する情報を見ても利用者を特定できない。以下では、トークン発行銀行とトークン償還銀行を区別せず、銀行と呼ぶ。なお、システムパラメータとして、素数位数 q の乗法群 $\mathbb{G}$ と生成元 $g \in \mathbb{G}$ が定められ、公開鍵の生成元 g が公開されているほか、銀行は予め秘密鍵-公開鍵ペア $(x, h = g^x)$ を計算し、公開鍵 h を全利用者に通知しているものとする。

【トークンの発行（図 5)】

- 銀行は、乱数 $w \in \mathbb{Z}_q^*$ を用いて $a = g^w$ を計算し、利用者に送る。
- 利用者は、乱数 $u, v \in \mathbb{Z}_q^*$ を用いて $a' = a^u g^v$ を計算する。
- 利用者は、$c' = \mathcal{H}(\mathtt{Serial}, a')$ と $c = c'/u$ を計算し、c を銀行に送る。なお、Serial は利用者が任意に選んだトークンのシリアル番号、$\mathcal{H}$ はハッシュ関数とする。
- 銀行は、秘密鍵 x を用いて $r = cx + w$ を計算し、利用者に送る[34]。
- 利用者は、$r' = ru + v$ を計算する。
- 利用者は、$(\mathtt{Serial}, a', r')$ をトークンとして保管する。

【トークンの使用】

- 利用者は、店舗にトークン $(\mathtt{Serial}, a', r')$ を送信する。
- 店舗は、$c' = \mathcal{H}(\mathtt{Serial}, a')$ を計算し、銀行の公開鍵 h に対して $g^{r'} = h^{c'} a'$ が成立すれば、トークンを受理する。成立しない場合には、取引を中断する。
- 店舗は銀行にトークンを送金する。銀行においても上記検証式が成立することを確認したうえで、トークンに見合いの現金を店舗の口座に入金する。

[33] 左辺 $= g^r = g^{cx+w} = h^c a =$ 右辺。

[34] 秘密鍵はトークンの金額に応じて用意されている。つまり、A 円分のトークンは x_A を用いて生成される。トークンを受け取った店舗は、h_A $(= g^{x_A})$ を用いて検証することでトークンの金額を確認することができる。

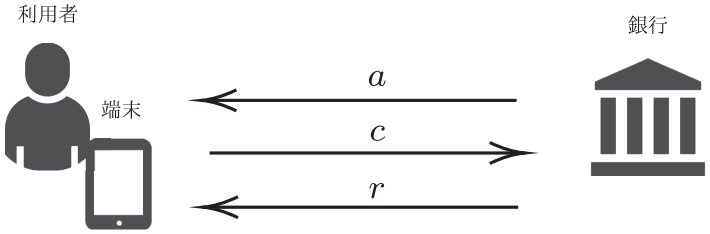


図 5: Schnorr ブラインド署名

Schnorr ブラインド署名（図 5）は、2 つの乱数 $u, v \in \mathbb{Z}_q^*$ によって、銀行から見える値 (a, c, r) をブラインド（ランダム化）された 3 つの値 (a', c', r') に変換する。この変換により、銀行は自分にとって既知の (a, c, r) から (a', c', r') を特定することはできない[35]。

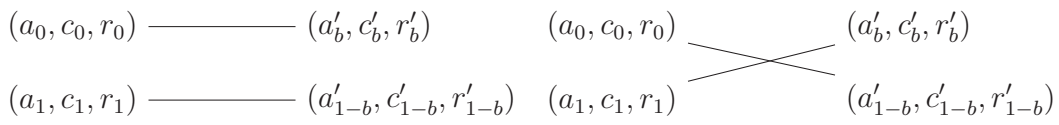
図 6 は銀行から見える値（銀行 View）と利用者が内部で計算する値（利用者 View）の違いを示したものである。銀行 View に現れる変数 (a, c, r) と、利用者 View に現れる変数 (a', c', r') は共に図 6 に示した検証式 (1)（署名の正当性の検証）を満たす正しい銀行の署名である。しかし、支払いに利用したトークンを特定可能なシリアル番号は銀行 View には含まれない。このため、銀行は償還されたトークンのシリアル番号をみても、どの利用者が使用したものであるかを知ることはできない。利用者 View から得られる署名のみがメッセージ `Serial` への銀行の署名[36]となり、これがシリアル番号 `Serial` をもつトークン $(\mathtt{Serial}, a', r')$ となる。最下段の 2 つの検証式は、銀行が自らの秘密鍵や乱数を用いて署名したものであることを検証する式（左）と、利用者が自ら発行した乱数を用いて銀行の署名をブラインドしたものであることを検証する式（右）に相当する。

銀行 View（x, wを秘密に保持)		利用者 View（u, vを秘密に保持)
$a = g^w$	$\rightarrow$	$a' = a^u g^v \;\; (= g^{uw+v})$
$c \;\; (= c'/u)$	$\leftarrow$	$c' = \mathcal{H}(\mathtt{Serial}, a')$
$r = cx + w$	$\rightarrow$	$r' = ru + v \;\; (= (cx + w)u + v)$
$g^r = h^c a$		$g^{r'} = h^{c'} a' \quad \left(\because g^{r'} = (g^x)^{cu} g^{uw+v}\right)$ (1)

図 6: 銀行 View と利用者 View の相違

[35] a, c, r とも $\mathbb{Z}_q$ の元であることから、いずれも q 通りの値をとる。従って、制約がなければ、(a, c, r) の組み合わせは最大 q^3 通りの値を取り得る。ただし、Schnorr 署名では、(a, c, r) の変数のうち 2 つが決まれば図 6 に示した検証式 (1) により、残りの変数が一意に決まるため、(a, c, r) が取り得る組み合わせは q^2 通りである。このとき、別の変数 $u, v \in \mathbb{Z}_q$ により、(a, c, r) から (a', c', r') が全単射で変換されるため、(a', c', r') が取り得る値も q^2 通りとなり、検証式を満たす全ての値をとりえる。すなわち、(a, c, r) から (a', c', r') を特定する情報は得られない。

[36] 利用者は c' ではなく c' を u で除した c を銀行に送り、銀行は c をメッセージハッシュ値として署名を生成するため、銀行は `Serial` を特定することができない。なお、c' の計算に $\mathcal{H}(\mathtt{Serial}, a')$ のように a' をハッシュ計算に加えるのは、プロトコルに従って a' を決めてから r' を求めさせるため (Feige, Fiat and Shamir, 1988) である。



(1) $b = 0$ のとき　　(2) $b = 1$ のとき

図 7: 匿名性を破れるか否かを試すゲーム

ロ　トークンの使用における匿名性

ここで、銀行（署名者）がトークンから利用者の情報を得ることができるかを考える。具体的には、銀行 $\mathrm{View}(a, c, r)$ と利用者 $\mathrm{View}(a', c', r')$ を 2 つずつ提示し、両者の対応関係を当てるゲームを行う。銀行の正解率が $1/2$ より高い場合には、トークンから元の署名の情報が洩れているということであり、銀行がトークンから利用者を特定できる可能性が生じる。

より具体的に説明すると以下のとおりである。銀行は出題者と共にトークン生成プロトコルを実行し、2 つの署名 (a_0, c_0, r_0), (a_1, c_1, r_1) を生成する。出題者は、その 2 つの署名に対応するトークンを銀行に提示するが、提示する順番はランダムに選んだ $b \leftarrow \{0, 1\}$ を用いて (a'_b, c'_b, r'_b), $(a'_{1-b}, c'_{1-b}, r'_{1-b})$ とする。銀行は提示されたトークンを見て、b が 0 であるか 1 であるかを推定する。

図 7 に示すように、正解が $b = 0$ の時は左側のストレートになり、正解が $b = 1$ の時は右側のクロスの対応になる。前述のように、乱数 $u, v \in \mathbb{Z}_q^*$ により、(a, c, r) は $(a' = a^u g^v, c' = cu, r' = ru + v)$ に写像され、検証式を満たす (a, c, r) の組全体への全単射となる。u, v の値に応じてストレートとクロスのいずれの対応もありえることから、銀行がこのゲームで勝利できる確率は、ランダムな推測確率である $1/2$ と僅かな差もつけられない。このことは、銀行がブラインド署名を用いて発行したトークンは暗号学的な意味での匿名性（unlinkability や blindness などと言う）が保たれることを意味している。

(4)　追跡可能 Schnorr ブラインド署名

Schnorr ブラインド署名により取引の匿名性を保つことができたが、二重支払いを行った利用者を特定するためには別途工夫が必要となる。トークンの発行体は利用者に預金口座を提供しており、利用者 ID を管理していると想定できるため、トークン発行時に利用者 ID を埋め込むことで二重支払いを行った利用者を特定することができる。本節で紹介する追跡可能 Schnorr ブラインド署名を利用したデジタル通貨は、取引の匿名性を保ちつつ、上記の課題を解決したものである (Brands, 1993)[37]。

システムパラメータとして、素数位数 q の乗法群 $\mathbb{G}$ と生成元 $g, g_1, g_2 \in \mathbb{G}$ が定められ、(g, g_1, g_2, q) が公開されているほか、銀行は予め秘密鍵-公開鍵ペア $(x, h = g^x)$ を計算し、公開鍵 h を全利用者に通知しているものとする。以下で示すように、生成元 g はトークンの生成（銀行の署名）に用いられ、生成元 g_1, g_2 は二重支払時の利用者の特定に用いられる。

[37] 二重支払いの防止については、トークン発行銀行で店舗からの還流（償還）時に過去の利用履歴の確認は可能であるが、店舗段階での確認は本方式では不可能である。次節のオブザーバー方式では、デジタル通貨ウォレットのデバイスにおいて検証可能であり、その仕組みを解説する。

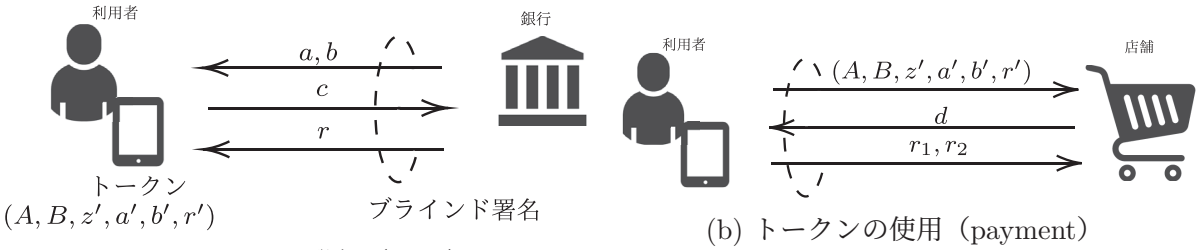
【セットアップ】

- 利用者は、トークンの発行要請ごとに乱数 $u_1 \in \mathbb{Z}_q^*$ を選び、$I = g_1^{u_1}$ を計算して銀行に送る。
- 銀行は、利用者が口座保持者であることが確認できれば、$z = (Ig_2)^x$ を利用者に送る。x は銀行の秘密鍵であり、z は利用者 ID を表す Ig_2 に対して銀行が発行した利用者証明書である。

【トークンの発行（図 8(a)）】

- 銀行は、乱数 $w \in \mathbb{Z}_q^*$ を用いて、$a = g^w$ と $b = (Ig_2)^w$ を計算し、a, b を利用者に送る。
- 利用者は、乱数 $s, u, v \in \mathbb{Z}_q^*$ と利用者 ID をブラインドした $A = (Ig_2)^s$ を用いて[38]、$z' = z^s, a' = a^u g^v, b' = b^{su} A^v$ を計算する。
- 利用者は、乱数 $x_1, x_2 \in \mathbb{Z}_q^*$ を用いて、$B = {g_1}^{x_1} {g_2}^{x_2}$ を計算する。さらに、$c' = \mathcal{H}(A, B, z', a', b')$ と $c = c'/u$ を計算し、c を銀行に送る。
- 銀行は、秘密鍵 x を用いて $r = cx + w$ を計算し、利用者に送る。
- 利用者は、$r' = ru + v$ を計算する。
- 利用者は、(A, B, z', a', b', r') をトークンとして保管する。

本方式では、生成元を g とした $h = g^x$ を銀行の公開鍵とするブラインド署名と、生成元を (Ig_2) とした利用者証明書である $z = (Ig_2)^x$ が銀行の公開共有する暗号鍵の生成に用いるデータを鍵とみなせることを利用した[39]ブラインド署名の 2 つが登場する。いずれも、ブラインド署名の対象となるメッセージは (A, B, z', a', b') であり、そのハッシュ値 c' を用いる。



(a) トークンの発行（issue）　　(b) トークンの使用（payment）

図 8: 二重支払時に追跡可能な Schnorr ブラインド署名

[38] $(Ig_2)^s = g_1^{u_1 s} g_2^s$ は、各生成元のべき $u_1 s$ と s の比 $u_1 s/s = u_1$ を一定に維持したまま、利用者 ID を表す Ig_2 を秘匿する。仮に、$(g_1^{u_1} g_2)^s = (g_1^{u_2} g_2)^{s'}$ を満たすような、別の利用者 ID $g_1^{u_2} g_2$ に偽装する u_2, s' を見つけられたとすると、システムパラメータ g_1, g_2 の間の離散対数 $\log_{g_1} g_2 = (u_1 s - u_2 s')/(s - s')$ を求めることができてしまう。これは離散対数問題を破ることになり矛盾である。すなわち、利用者がブラインド後の利用者 ID である $A = (Ig_2)^s$ に対して、g_1, g_2 のそれぞれのべきである $u_1 s$, s を知っている状態でいるためには、素直に利用者 ID である Ig_2 を s 乗してブラインドする以外に方法がないことを意味する。次のステップで計算する $B = g_1^{x_1} g_2^{x_2}$ と併せて、支払い時に店舗に対して g_1 と g_2 の生成元のそれぞれで個別に証明させることで、利用者が $u_1 s$, s を知っていることを店舗が検証する。

[39] 正確には、銀行は離散対数 $x = \log_g h$ と離散対数 $x = \log_{(Ig_2)} z$ の 2 つの値を“知っていること”かつ同じ値 (x) であることを、x を明かさずに証明（Proof of Knowledge）することを利用した 2 つのブラインド署名、である。

前者は公開鍵 h に対するブラインド署名 (h, a', r')、後者は z' を公開鍵にみたてたブラインド署名 (z', b', r') となっている（表 2 の二行目利用者 view を参照)。いずれも c' と r' が共通して用いられている。ブラインドあり（銀行 View）とブラインドなし（利用者 View）を合わせると、以下の 4 つの検証式が成立する。

表 2: 銀行が発行する 2 つのブラインド署名

	生成元g, 公開鍵h	生成元Ig_2, 利用者証明書z
銀行 View	$g^r = h^c a$	$(Ig_2)^r = z^c b$
利用者 View	$g^{r'} = h^{c'} a'$	$A^{r'} = z'^{c'} b'$

表 3: ブラインド署名における銀行 View と利用者 View の対応

銀行 View（x, wを秘密に保持）		利用者 View（s, u, vを秘密に保持）
$z = (Ig_2)^x$	$\longrightarrow$	$z' = z^s \ \ (= (Ig_2)^{sx} = A^x)$
$b = (Ig_2)^w$	$\longrightarrow$	$b' = b^{su} A^v \ \ (= (Ig_2)^{s(uw+v)} = A^{uw+v})$
$c \ \ (= c'/u)$	$\longleftarrow$	$c' = \mathcal{H}(A, B, z', a', b')$
$r = cx + w$	$\longrightarrow$	$r' = ru + v \ \ (= (cx + w)u + v)$
$(Ig_2)^r = z^c b$		$A^{r'} = z'^{c'} b'$ (2)
		$\left(\because A^{r'} = A^{(cx+w)u+v} = z'^{cu} A^{uw+v} = z'^{c'} b'\right)$

【トークンの使用（図 8(b))】

- 利用者は、店舗にトークン (A, B, z', a', b', r') を送信する。
- 店舗は、乱数 $d \in \mathbb{Z}_q^*$ を利用者に送る。
- 利用者は、$r_1 = d(u_1 s) + x_1$ と $r_2 = ds + x_2$ を計算し、r_1 と r_2 を店舗に送る。
- 店舗は、$c' = \mathcal{H}(A, B, z', a', b')$ を計算し、$g^{r'} = h^{c'} a', A^{r'} = z^{c'} b'$（銀行の署名）と $g_1^{r_1} g_2^{r_2} = A^d B$（利用者 ID の追跡可能性の証明）が成立すれば、トークンを受理する。成立しない場合には、取引を中断する。
- 店舗は銀行にトークンを送信する。銀行においても上記の署名検証式が成立し、かつ過去に使用されたことのないトークンであることが確認できれば、トークンに見合いの現金を店舗の口座に入金する。

銀行 View における (z, b, c, r) と利用者 View における (z', b', c', r') は、いずれも Schnorr ブラインド署名と同じ $(c, r), (c', r')$ で検証式 (2) を満たすところがポイントである。Schnorr ブラインド署名との違いは、利用者 View における (z', b', c', r') が乱数 (s, u, v) でブラインドされている点であり、これにより利用者の ID をランダム推測以上に効率よく推定することはできないようになっている。

【二重支払者の特定】

次に、二重支払いされたトークンが銀行に還流されたとき、その 2 つのトークンから利用者の ID を特定する手順を説明する。なお、二重支払いでは、同じトークン (A, B, z', a', b', r') が使用されるが、トークン使用時に店舗から発行された 2 つの $d \neq d' \in \mathbb{Z}_q$ に対して、検証式を満たす r_1, r_2, r_1', r_2' が発行されている状態となる。ここで、d はトークンが使われた初回

時に店舗で発行された乱数であり、d' は同一トークンの二度目の利用において店舗で発行された乱数である。

- 銀行は、トークン (A, B, z', a', b', r') と、支払いに使用された (d, r_1, r_2) と (d', r_1', r_2') を入手する。
- 銀行は、$u_1 = (r_1 - r_1')/(r_2 - r_2')$ を計算し、登録時の $I = g_1^{u_1}$ と照合して利用者を特定する。

なお、r_1, r_2, r_1', r_2' に対しては、4 つの未知数 u_1, s, x_1, x_2 に対して、以下の 4 つの式が $\mathbb{Z}_q$ 上で成立していることから、式を解いて $u_1 = (r_1 - r_1')/(r_2 - r_2')$ を求めることが可能である。

$$r_1 = d\,(u_1 s) + x_1, \qquad r_2 = ds + x_2 \tag{3}$$
$$r_1' = d'\,(u_1 s) + x_1, \qquad r_2' = d's + x_2 \tag{4}$$

以上により、ブラインド署名により利用者の匿名性を確保しつつ、二重支払いが行われた際には利用者の ID を特定できるスキームが構築できる。しかし、これは不正を働いた利用者の ID を事後的に特定できるというものであり、不正そのものを防止するスキームとはなっていない。できれば、二重支払いを未然に防止できることが望ましい。

(5) オブザーバー方式の構成

オブザーバーは、銀行から利用者（端末）に供与された耐タンパーデバイスであり、利用者の匿名性を確保しながら、銀行が利用者に求めるルール（ここでは二重支払いの禁止）を遵守させることを目的としている。オブザーバーには、前節までの追跡可能 Schnorr ブラインド署名の仕組みに加えて、デバイス中のオブザーバーが関与する署名の仕組みが追加される。具体的には、利用者からトークン発行の依頼を受けた銀行は利用者 ID を埋め込んだトークン（の元になる情報）を利用者に送り返す際に、利用者の ID と利用者のオブザーバーの ID の両方を埋め込む。これにより、支払いの際にオブザーバーの関与が必要となる。さらに、オブザーバーに一度しか支払いに関与しないようプログラムしておくことにより、二重支払いを未然に防ぐことができる。また、万が一、オブザーバーの耐タンパー性が崩れ[40]、オブザーバーの秘密鍵が漏洩したことにより二重支払いを店舗が未然に検出することが不可能になったとしても、前出のケースと同様、銀行への償還段階での署名検証により二重支払いした利用者を特定することは可能である。

システムパラメータとして、素数位数 q の乗法群 $\mathbb{G}$ と生成元 $g, g_1, g_2 \in \mathbb{G}$ が定められ、(g, g_1, g_2, q) が公開されているほか、銀行は予め秘密鍵-公開鍵ペア $(x, h = g^x)$ を計算し、公開鍵 h を全利用者に通知しているものとする。

【オブザーバーのセットアップ（図 9）】

- 利用者は、乱数 $u_1 \in \mathbb{Z}_q^*$ を選び、$g_1^{u_1}$ を計算して銀行に送る。
- 銀行は、利用者が口座保持者であることが確認できれば、次の手順でオブザーバーをセットアップする。
- オブザーバーは、内部で乱数 $o_1 \in \mathbb{Z}_q^*$ を選び、$A^{\mathcal{O}} = {g_1}^{o_1}$ を計算して $A^{\mathcal{O}}$ を銀行に送る。
- 銀行は、$I = A^{\mathcal{O}} g_1^{u_1}$ を計算し、$z = (I g_2)^x$ を利用者に送る。

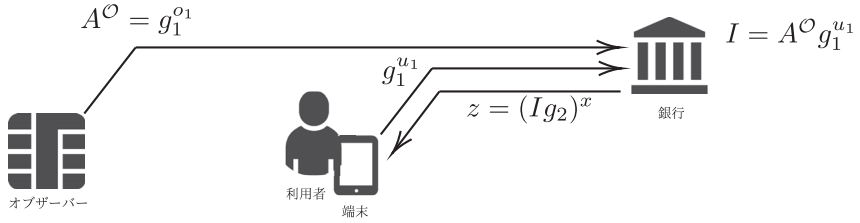


図 9: オブザーバーのセットアップ。

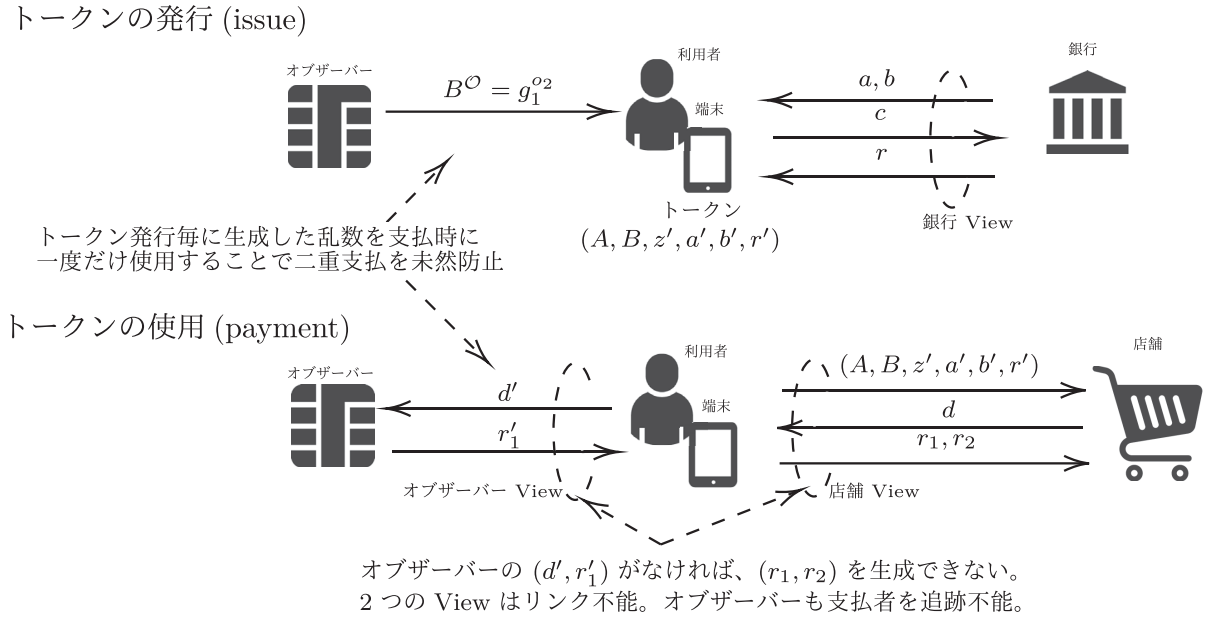


図 10: オブザーバー方式における匿名性と二重支払防止（透明性）の両立

【トークンの発行（図 10）】

- 銀行は、秘密鍵 x と乱数 $w \in \mathbb{Z}_q^*$ を用いて、$a = g^w$、$b = (Ig_2)^w$ を計算し、(a, b) を利用者に送る。
- 利用者は、乱数 $s, u, v \in \mathbb{Z}_q^*$ を用いて、$z' = z^s, A = (Ig_2)^s$, $a' = a^u g^v$, $b' = b^{su} A^v$ を計算する。
- 利用者は、オブザーバーから $B^{\mathcal{O}} = {g_1}^{o_2}$ を受け取る。なお、$o_2 \in \mathbb{Z}_q^*$ は取引ごとにオブザーバーが選んだ乱数である。
- 利用者は、乱数 $x_1, x_2, e \in \mathbb{Z}_q^*$ を用いて、$B = g_1^{x_1} g_2^{x_2} \cdot (A^{\mathcal{O}})^{es} B^{\mathcal{O}}$ を計算する。さらに、$c' = \mathcal{H}(A, B, z', a', b')$ と $c = c'/u$ を計算し、c を銀行に送る。
- 銀行は、$r = cx + w$ を計算し、利用者に送る。
- 利用者は、$r' = ru + v$ を計算する。
- 利用者は、(A, B, z', a', b', r') をトークンとして保管する。

[40]耐タンパーデバイスから秘密鍵を漏洩させるには前出の AVA_VAN.5 等のペネトレーションテストを凌ぐ攻撃を実施しなければならず、通常は困難であると考えられている。

【トークンの使用（図 10)】

- 利用者は、店舗にトークン (A, B, z', a', b', r') を送信する。
- 店舗は、乱数 $d \in \mathbb{Z}_q^*$ を利用者に送る。
- 利用者は、乱数 $e \in \mathbb{Z}_q^*$ を生成し、$d' = s(d + e)$ をオブザーバーに送る。
- オブザーバーは、$r_1' = d'o_1 + o_2$ を利用者に送る。
- 利用者は、$r_1 = r_1' + d(u_1 s) + x_1$ と $r_2 = ds + x_2$ を計算し、r_1 と r_2 を店舗に送る。
- 店舗は、$c' = \mathcal{H}(A, B, z', a', b')$ を計算し、$g^{r'} = h^{c'}a', A^{r'} = z'^{c'}b'$（銀行の署名の検証）と $g_1^{r_1}g_2^{r_2} = A^d B$（トークンの追跡可能性の証明）が成立すれば、トークンを受理する。成立しない場合には、取引を中断する。
- 店舗は銀行にトークンを送金する。銀行においても上記の署名検証式が成立し、かつ過去に使用されたことのないトークンであれば、トークンに見合いの現金を店舗の口座に入金する。

オブザーバー方式は、追跡可能 Schnorr ブラインド署名を応用したものであるが、利用者の ID を表す I の構成が異なる。これは、オブザーバーが生成した情報 $A^{\mathcal{O}} = g_1^{o_1}$ が I に追加され、$I = g_1^{u_1+o_1}$ の形式となるためである。ここで、u_1 は利用者 ID を示す $g_1^{u_1}$ の秘密鍵であり、o_1 はオブザーバー ID を示す $A^{\mathcal{O}}$ の秘密鍵になっており、2 つの ID を合成した $I = g_1^{u_1+o_1}$ に関する秘密鍵は、利用者とオブザーバーの両者が協力しなければ合成できない構造になっていることに注意して欲しい。これに伴い、下表のように A, B, z' の値も変更される。

	追跡可能 Schnorr ブラインド署名		オブザーバー方式
I	$g_1^{u_1}$	$\rightarrow$	$A^{\mathcal{O}}g_1^{u_1}$
A	$(Ig_2)^s = (g_1^{u_1}g_2)^s$	$\rightarrow$	$(Ig_2)^s = (A^{\mathcal{O}}g_1^{u_1} \cdot g_2)^s$
B	$g_1^{x_1}g_2^{x_2}$	$\rightarrow$	$g_1^{x_1}g_2^{x_2} \cdot (A^{\mathcal{O}})^{es}B^{\mathcal{O}}$
z	$(Ig_2)^x = (g_1^{u_1}g_2)^x$	$\rightarrow$	$(Ig_2)^x = (g_1^{u_1}g_2 \cdot g_1^{o_1})^x$

トークン受領時における店舗側の検証式は、次のように確認できる。

$$\begin{aligned} g_1^{r_1}g_2^{r_2} &= g_1^{r_1'+d(u_1 s)+x_1}g_2^{ds+x_2} \\ &= g_1^{(d'o_1+o_2)}g_1^{d(u_1 s)}g_2^{ds}g_1^{x_1}g_2^{x_2} \\ &= (A^{\mathcal{O}})^{d'}B^{\mathcal{O}}(g_1^{u_1})^{ds}g_2^{ds}g_1^{x_1}g_2^{x_2} \\ &= (A^{\mathcal{O}})^{s(d+e)}B^{\mathcal{O}}(g_1^{u_1})^{ds}g_2^{ds}g_1^{x_1}g_2^{x_2} \\ &= (A^{\mathcal{O}}g_1^{u_1}g_2)^{ds}(A^{\mathcal{O}})^{se}B^{\mathcal{O}}g_1^{x_1}g_2^{x_2} \\ &= A^d B \end{aligned} \tag{5}$$

【二重支払者の特定】

- 銀行は、トークン (A, B, z', a', b', r') と、支払いに使用された (d, r_1, r_2)、および新たな支払いに使われようとしている (d', r_1', r_2') を入手する。
- 銀行は、$o_1 + u_1 = (r_1 - r_1')/(r_2 - r_2')$ を計算し、$g_1^{u_1+o_1}$ とセットアップ時の $I = A^{\mathcal{O}}g_1^{u_1}$ と照合して利用者を特定する。

二重支払いの際には、追跡可能Schnorrブラインド署名と同様、1つのトークン (A, B, z', a', b') に対して、2つの店舗による2つの異なるチャレンジ d, d' に対応する以下の値 (r_1, r_2), (r'_1, r'_2) が銀行に還流してくることから、下記2つの式により、$o_1 + u_1 = (r_1 - r'_1)/(r_2 - r'_2)$ が計算できる。これを使って $g_1^{u_1+o_1}$ を求めることにより、セットアップ時の $I = A^{\mathcal{O}} g^{u_1}$ と照合して利用者を特定することができる。

$$r_1 = s(d+e)o_1 + o_2 + d\,(u_1 s) + x_1, \qquad r_2 = ds + x_2 \tag{6}$$

$$r'_1 = s(d'+e)o_1 + o_2 + d'\,(u_1 s) + x_1, \qquad r'_2 = d's + x_2 \tag{7}$$

上記のとおり、オブザーバー方式は、耐タンパーデバイスを利用者端末に搭載させることで、1) 利用者に二重支払いさせないよう強制するほか、2) 仮に耐タンパー性が崩れて二重支払いが行われたとしても、IDの追跡が可能なスキームになっている。仮に、銀行が不正なオブザーバーを導入し、利用者のIDを不正に特定しようとしても、オブザーバーのViewと店舗のViewは完全に切り離されていることから、オブザーバーのViewから支払先を特定することはできない（図10参照）。すなわち、利用者の匿名性は、利用者が端末を適切に運用する限り暗号学的に保護される。

4　実行証跡付セキュアプロセッサによる柔軟な匿名性・透明性

前節で整理したオブザーバー方式は、デジタル通貨の二重支払いを行わないよう利用者の行動を強制する方式であった。これに対し、本節で紹介する実行証跡付セキュアプロセッサ (Pass, Shi and Tramér, 2017) は、任意のプログラムを耐タンパーデバイスで実行させる方式に関する理論である。近年、TEEなどの隔離実行領域が利用可能になってきたことや、GP-SEやTPM（Trusted Platform Module）などのプログラマブル・セキュアエレメントが普及してきたことを受けて、それらの安全性を学術的に取り扱うために整備された。

(1)　概要

イ　仮定

実行証跡付セキュアプロセッサについては、最新の耐タンパーデバイスに一定の仮定をもたせることで、後述の「内部記憶を持つプログラムの難読化」を実現可能であることが証明されている (Pass, Shi and Tramér, 2017)。これは既存のTEEやGP-SEを実行証跡付セキュアプロセッサと見做すことができれば、銀行がオブザーバーを難読化ソフトウェアとして配布し、利用者が自分のデバイスにダウンロードして実行することで、銀行が要請する二重支払防止機能を利用者のデバイスで確実に実行させることができることを示している。本節では、実行証跡付セキュアプロセッサに求められる仮定について述べる。

1. 耐タンパー性を有するセキュアプロセッサが端末に具備されている。

   耐タンパー性は、サイドチャネル攻撃等の非侵襲型攻撃や、リバースエンジニアリングやフォールトインジェクション攻撃等の侵襲型攻撃に対して、内部に保存された暗号鍵や機密情報を外部に漏洩しないという性質である。理論的に安全な構成方法は知られて

いないが、現実的には、FIPS 140[41]やCC認証（ISO/IEC 15408）におけるAVA_VAN.5等の網羅的なペネトレーションテストを通過したものを「耐タンパー性を有する」と呼ぶことが多い。

2. セキュアプロセッサにアクセス可能なプログラム／デバイスが限定されている。

   セキュアプロセッサは内部にホワイトリスト reg を持ち、セキュアプロセッサにアクセス可能なプログラム／デバイス $\mathcal{P}$ を識別する機能があると仮定する (Pass, Shi and Tramér, 2017)。具体的には、2節 (2) ロ で述べたコード署名が付されたスマートフォンアプリおよびスマートフォンデバイスが $\mathcal{P}$ に、ARA-C に登録されたコード署名検証用の公開鍵データの集合が reg に該当する。

3. 外部からプログラムを導入でき、実行結果には必ず証跡（attestation）が付与される。

   本仮定は、実行証跡付セキュアプロセッサに特有のものとなる。現在の TEE や GP-SE 等の多くのセキュアプロセッサは外部からプログラムを導入可能である。また、これらのセキュアプロセッサが必ずしも実行結果に証跡を付与しているとは限らないが、導入するプログラムの構成を工夫することで容易に証跡（実行証跡、実行結果に対するデジタル署名）を付与することができる (Pass, Shi and Tramér, 2017)。

4. 匿名で実行証跡が付与される（Anonymous Attestation）。

   前述の実行証跡は、セキュアプロセッサに関する匿名性を保った状態で署名できることを仮定する。すなわち、実行証跡からセキュアプロセッサを特定することはできないと仮定する。現実には、不正なデバイスによる実行証跡を特定できるよう失効リスト（revocation list）を配布することが求められるため、セキュアプロセッサの匿名性についてはより詳細な議論が必要である。Intel SGX 等では特殊なグループ署名 (Brickell and Li, 2010) を導入し[42]、実行証跡の匿名性と不正デバイスが出力した実行証跡の失効を両立させる方式を実装している。ここでは議論を単純化するために、全セキュアプロセッサが共通の公開鍵 mpk と秘密鍵 msk を共有し、デジタル署名に用いた鍵からセキュアプロセッサを特定できないと仮定し、不正デバイスの追跡等には言及しないこととする。

実行証跡付セキュアプロセッサの仮定は、特に 1. と 2. が強い仮定となるが、市場に広く普及している TEE や GP-SE などのデバイスは 2 節 (2) ロ で述べたように安全性や機能面の拡張が進んで来ており、十分とは言えないまでも、実行証跡付セキュアプロセッサに必要な仮定をほぼ満たしている。

ロ　理想機能

上記を仮定した際に想定される実行証跡付セキュアプロセッサの理想機能 $\mathcal{G}_{\mathrm{att}}$ を定義する。ここで、Σ は仮定 3. と 4. に対応するデジタル署名アルゴリズムであり、署名生成用の鍵

[41] FIPS は Federal Information Processing Standards の略。FIPS140 は、米国連邦政府の暗号モジュールが満たすべきセキュリティ要件を記載した政府調達基準。

[42] Direct Anonymous Attestation（DAA）(Brickell, Camenisch and Chen, 2004) もしくは Enhanced Privacy ID（EPID）(Brickell and Li, 2010) と呼ばれ、活発に研究されている。DAA や EPID はグループ署名の一種である。特定グループの誰かが署名したことは確認できるが、誰かは特定できないというグループ署名の標準的な特徴に加え、グループ署名の管理者であっても署名から署名者を特定できないという性質を有する。この性質により、チップ製造者であってもプライバシーを侵害できない。

msk によるメッセージ m の署名は $\Sigma.\mathrm{Sign}_{\mathrm{msk}}(m)$ と表される[43]。また、reg は、$\mathcal{G}_{\mathrm{att}}$ を使用可能なプラットフォーム $\mathcal{P}$ の全体集合 ($\mathcal{P} \in \mathtt{reg}$) と定義する[44]。

$\mathcal{G}_{\mathrm{att}}$ は以下の 4 つのインターフェイスを持つ (図 11(a))。最初の 2 つはどのプラットフォームでも利用可能なインターフェイスだが、残りの 2 つは特定のプラットフォーム $\mathcal{P} \in \mathtt{reg}$ のみが利用可能な制限されたインターフェイスである。

1. 初期化 `initialize`

   実行証跡に用いるデジタル署名の秘密鍵と公開鍵のペア (msk, mpk) を生成し、内部記憶 T を空集合 $\varnothing$ に初期化する。

2. 署名検証鍵の取得 `getpk`()

   実行証跡の署名検証用の鍵 mpk を出力する。

3. プログラムの登録 $\mathtt{install}(idx, \mathtt{prog})$ from some $\mathcal{P} \in \mathrm{reg}$

   プログラム prog[45]を実行証跡付セキュアプロセッサに登録し、prog に固有のメモリ空間 mem を割り当てる。prog を登録する際のセッション ID である idx に対して、(idx, prog, mem) の組をエンクレーブ（enclave）と呼ぶ。エンクレーブの識別子 eid をランダムに生成し、プラットフォーム $\mathcal{P}$ 毎に区別して内部記憶 T に登録する。

   $$T[eid, \mathcal{P}] := (idx, \mathtt{prog}, \overrightarrow{0})$$

4. プログラムの実行 $\mathtt{resume}(eid, \mathtt{inp})$ from some $\mathcal{P} \in \mathrm{reg}$

   eid, $\mathcal{P}$ で指定されるエンクレーブに格納された prog を、入力 inp に対して実行し、その結果 outp を以下の形式の実行証跡として出力する。

   $$(idx, eid, \mathtt{prog}, \mathtt{outp}, \sigma)$$

   ここで σ は実行証跡付セキュアプロセッサの実行証跡であり、以下のデジタル署名アルゴリズムで計算される。

   $$\sigma \leftarrow \Sigma.\mathrm{Sign}_{\mathrm{msk}}(idx, eid, \mathtt{prog}, \mathtt{outp})$$

実行証跡付セキュアプロセッサが実行されると、その実行結果には必ず outp とそのデジタル署名 σ が含まれる。デジタル署名により、outp が実行証跡付セキュアプロセッサ $\mathcal{G}_{\mathrm{att}}$ で実行されたことが第三者によって検証可能となる。

---

[43]デジタル署名アルゴリズム Σ は存在的偽造不能性（EUF-CMA）を満たすと仮定している。存在的偽造不能性は、デジタル署名に関する最強の安全性である。具体的には、セキュリティパラメータ（例えば鍵長など）の多項式で表される実行ステップ数に制限された（署名用の秘密鍵を知らない）攻撃者が、自由に選んだメッセージの署名を得られる条件下（署名オラクルへのアクセスが許される条件下）で、署名検証用の公開鍵の情報と多数のメッセージと署名の組から得られる情報を用いて、それらに含まれない新しい署名を自らの力で偽造できる確率が無視できるほど小さいという性質を言う。また、署名検証アルゴリズム $\Sigma.\mathrm{Vf}_{\mathrm{mpk}}(\sigma)$ は署名検証鍵（公開鍵）mpk による署名 σ の検証式を表し、署名が正しければ 1 を出力し、そうでなければ 0 を出力する。

[44]SGX 等が搭載された PC や Android 端末等で実行される一般のプログラムをプラットフォーム $\mathcal{P}$ と呼ぶ。実用上は、2 節 (2) ロ．で述べたコード署名とアクセス制御機構でアクセス許可された正規アプリケーションプログラムがプラットフォーム $\mathcal{P}$ に対応し、正規アプリケーションプログラム全体の集合が reg に対応する。

[45]セキュアプロセッサ内で実行するプログラム。TEE ではそれぞれのプロセッサ用のバイナリコードであり、GP-SE 等の耐タンパーデバイスでは Java アプレットが想定される。

$\mathcal{G}_{\mathrm{att}}[\Sigma, \mathtt{reg}]$

On initialize
| $(\mathrm{mpk},\ \mathrm{msk}\ ) := \Sigma.\mathrm{KeyGen}\left(1^{\lambda}\right), \quad T = \emptyset;$
On receive `getpk`() *from some* $\mathcal{P}$
| send mpk to $\mathcal{P}$;

エンクレーブ処理

On receive `install`$(idx, \mathtt{prog}$ *from some* $\mathcal{P} \in \mathtt{reg})$
| $\mathcal{P}$ が honest の時は、$idx = sid$
| $eid \in \{0,1\}^{\lambda}$; // エンクレーブ `ID` をランダムに生成
| $T[eid, \mathcal{P}] := (idx, \mathrm{prog}, \overrightarrow{0})$; // `mem` を $\vec{0}$ に初期化し格納
| eid を $\mathcal{P}$ に送信;
On receive `resume`$(\ eid, \mathtt{inp}$ *from some* $\mathcal{P} \in \mathtt{reg})$
| $(idx, \mathtt{prog}, \mathtt{mem}) := T[eid, \mathcal{P}]$; // $[eid, \mathcal{P}] \notin T$ なら停止
| $(\mathtt{outp}, \mathtt{mem}) := \mathtt{prog}(\mathtt{inp}, \mathtt{mem})$; // `prog` を実行
| $T[eid, \mathcal{P}] := (idx, \mathtt{prog}, \mathtt{mem})$; // 内部状態 T を更新
| $\sigma := \Sigma.\mathrm{Sig}_{\mathrm{msk}}(idx, eid, \mathtt{prog}, \mathtt{outp})$; // 証跡 σ を生成
| $(\mathtt{outp}, \sigma)$ を $\mathcal{P}$ に送信;

(a) 実行証跡付セキュアプロセッサの理想機能 $\mathcal{G}_{\mathrm{att}}$

$\mathtt{prog}_{\mathrm{obf}}$

On input $(\textit{“keyex”}, g^a)$*:*
| $b \in \mathbb{Z}_p$; // b をランダムに選ぶ
| store$sk := (g^a)^b$; // 秘密鍵 sk を計算して `mem` に格納
| **return** $\left(g^a, g^b\right)$;
On input $(\textit{“obfuscate”}, ct)$*:*
| $f := \mathtt{AE.Dec}_{sk}(ct)$; // 暗号文 ct から f を復号
| store$(f, \mathtt{st} := \perp)$; // f と f の状態 `st` を初期化して格納
| **return** OK;
On input $(\textit{“compute”}, x, y')$*:*
| $(f, \mathtt{st})$ を取り出す;
| **if** $y' \neq \perp$ **then**
| | **return** y'
| **else**
| | $(y, \mathtt{st}') := f(x, \mathtt{st})$; // f を計算
| | $\mathtt{st} := \mathtt{st}'$; // 内部状態 `st` を更新
| **return** y

(b) $\mathtt{prog}_{\mathrm{obf}}$ の機能

図 11: 実行証跡付セキュアプロセッサの理想機能 $\mathcal{G}_{\mathrm{att}}$ と内部記憶を持つプログラムの難読化

(2) 内部記憶のあるプログラムの難読化（Stateful Obfuscation）

イ 概要

難読化 (Barak et al., 2001; Goldwasser and Rothblum, 2007) とは、与えられたプログラムを、解析が困難なプログラムに変換することである。理想的に難読化されたプログラムは、手元のプロセッサで実行されているにも関わらず、あたかもブラックボックスの中で計算されているかのように、内部の計算過程からは何も有益な情報が得られない。そのため、攻撃者はブラックボックスの外部から観測できる入力と出力だけから内部の処理を推定する外ない状況に置かれる。このような性質を達成する理想的な難読化を、仮想ブラックボックス難読化 (Barak et al., 2001) と呼ぶ。仮想ブラックボックス難読化されたプログラムによる計算は、手元のプロセッサで実現できる秘密計算になっている。

例えば、プログラム内部に鍵がハードコードされているケースについて、入力された暗号文を内部で復号し、復号されたデータを用いて行った計算結果を再暗号化して出力するプログラム f を考える。入出力は暗号化されているため、適切な暗号化アルゴリズムが用いられていれば、入出力からは秘密計算の内容を推測することは困難である。しかし、内部の計算過程がホワイトボックスで、f のプログラムコードにアクセスできる攻撃者であれば、鍵を抽出することで内部で行われている処理を知ることができる。仮想ブラックボックス難読化が実現できれば、f の内部処理がブラックボックス化されることにより、内部にハードコードされた鍵が取り出されないことが保証され、プログラム f は安全に秘密計算される。

さらに、内部記憶を持つプログラムの難読化とは、データベースのように内部に記憶を持つプログラムに、難読化の概念を拡張したものであり、内部記憶とプログラムコードの両方にアクセスできる攻撃者に対しても、入出力から得られる以上の情報を攻撃者に与えないものをいう。前述の秘密計算は、内部に記憶を持たない関数的な計算を想定していたが、内部記憶を持つプログラム（以下、履歴依存プログラム）の仮想ブラックボックス難読化は、内部にデータベース等の記憶を持ち、入力履歴に応じて出力値が変化するタイプのプログラムに対しても、プログラムにハードコードされた鍵や内部記憶の内容を取り出されることなく、あたかもブラックボックス内部で実行されているかのように攻撃者から見える難読化を指す。

仮想ブラックボックス難読化は、ソフトウェアのみでは実現できないことが理論的に証明されている (Barak et al., 2001)。そのため、難読化の定義を緩和したり、破壊できない耐タンパー技術があると仮定を追加するなどの対応により達成可能な難読化を模索している (例えば、Goldwasser and Rothblum, 2007)。

ロ 内部記憶を持つプログラムの難読化の構成

サーバ $\mathcal{S}$ とクライアント $\mathcal{C}$ がそれぞれ $\mathcal{G}_{\mathrm{att}}$ を具備している状況を考える。$\mathcal{C}$ が履歴依存プログラム f を難読化して $\mathcal{S}$ に送付し、$\mathcal{S}$ に f を実行させる状況を想定する。目標は、$\mathcal{S}$ に f の入出力から得られる情報以上の情報を与えないことである。

難読化の構成は、図 12 に示すように（a）登録と（b）実行の二段階に分かれている。図 12 に記載している $\mathtt{prog}_{\mathrm{obf}}$ の各関数を集約したものが図 11(b) であり、この $\mathtt{prog}_{\mathrm{obf}}$ を $\mathcal{S}$ の $\mathcal{G}_{\mathrm{att}}$ に `install` することで、任意の履歴依存プログラム f の難読化が実現される。

（イ）難読化した履歴依存プログラム f の登録 $\mathcal{S}$ に対する f の難読化は、$\mathcal{C}$ と $\mathtt{prog}_{\mathrm{obf}}$ 間で鍵を共有し、共有した鍵で関数 f を暗号化して送信のうえ、$\mathtt{prog}_{\mathrm{obf}}$ に登録されることで実施される。関数 f の登録の処理は、大きくわけると、$\mathcal{S}$ の $\mathcal{G}_{\mathrm{att}}$ に $\mathtt{prog}_{\mathrm{obf}}$ をインストール（以

下の①∼③)、$\mathcal{C}$ と $\mathtt{prog}_{\mathrm{obf}}$ 間での暗号鍵 sk の共有（以下の④∼⑤)、sk による f の暗号化と $\mathtt{prog}_{\mathrm{obf}}$ への送信（⑥、⑦)、$\mathtt{prog}_{\mathrm{obf}}$ における f の登録（⑧）からなる。

① $\mathcal{G}_{\mathrm{att}}$ から $\mathcal{C}$ に難読化実行のメッセージを送信：利用者は、自分の端末（$\mathcal{C}$）内の理想機能（$\mathcal{G}_{\mathrm{att}}$）を起動し、$\mathcal{G}_{\mathrm{att}}$ から（$\mathcal{C}$ にインストールされている）$\mathcal{S}$ と交信するためのアプリに対して、「関数 f を難読化せよ」とのメッセージ (“obfuscate”, f) を送信する。$\mathcal{C}$ 内のアプリがこのメッセージを受け取ると、$a \in \mathbb{Z}_p$ とセッション識別子 $sid \in \mathbb{Z}$ をランダムに生成する[46]。

② $\mathcal{S}$ と共有する暗号鍵の生成に用いるデータを送信：$\mathcal{C}$ から $\mathcal{S}$ に sid と「g^a を入力として鍵交換せよ」とのメッセージ $(sid, (\text{“}keyex\text{”}, g^a))$ を送信する。

③ $\mathcal{S}$ の $\mathcal{G}_{\mathrm{att}}$ に $\mathtt{prog}_{\mathrm{obf}}$ をインストール：$\mathcal{S}$ に具備された実行証跡付セキュアプロセッサである $\mathcal{G}_{\mathrm{att}}$ で $\mathtt{install}(sid, \mathtt{prog}_{\mathrm{obf}})$ を実行する。これにより、$\mathcal{G}_{\mathrm{att}}$ 内部に $\overrightarrow{0}$ に初期化されたメモリ領域が確保され、エンクレーブ $(sid, \mathtt{prog}_{\mathrm{obf}}, \overrightarrow{0})$ が生成される。このエンクレーブに対応する識別子 eid がランダムに生成され、サーバ $\mathcal{S}$ が利用可能なエンクレーブとして、$\mathcal{G}_{\mathrm{att}}$ の記憶領域 T に登録される。

$$T[eid, \mathcal{S}] := \left(sid, \mathtt{prog}_{\mathrm{obf}}, \overrightarrow{0}\right)$$

④ $\mathtt{prog}_{\mathrm{obf}}$ によって暗号鍵の生成に用いるデータと実行証跡を $\mathcal{C}$ に送信：$\mathcal{S}$ の $\mathcal{G}_{\mathrm{att}}$ 間で $\mathtt{resume}(eid, (\text{“keyex”}, g^a))$ を実行すると、$\mathcal{G}_{\mathrm{att}}$ のエンクレーブ処理で $\mathtt{prog}_{\mathrm{obf}}(\text{“}keyex\text{”}, g^a)$ が実行される。$\mathtt{prog}_{\mathrm{obf}}$ は、入力 (“keyex”, g^a) が与えられると、内部で乱数 $b \in \mathbb{Z}_p$ を生成し、Diffie-Hellman 鍵

$$sk := (g^a)^b$$

を計算して内部に記憶（store）し[47]、(g^a, g^b) を出力する[48]。この値には $\mathcal{G}_{\mathrm{att}}$ による実行証跡として署名 σ が施され、計算に用いた $\mathtt{prog}_{\mathrm{obf}}$ や sid, eid と共に以下の形式で出力される。

$$\left(sid, eid, \mathtt{prog}_{\mathrm{obf}}, \left(g^a, g^b\right), \sigma\right) \tag{8}$$

⑤ $\mathcal{G}_{\mathrm{att}}$ から実行証跡を検証する鍵を入手：$\mathcal{C}$ は実行証跡を検証するために、$\mathcal{C}$ の $\mathcal{G}_{\mathrm{att}}$ から検証鍵 mpk を取得する。

⑥ 実行証跡を検証し関数を暗号化：mpk を用いてステップ④で出力される実行証跡 σ を検証し、①で生成した a を用いて Diffie-Hellman 鍵

$$sk := (g^b)^a$$

を計算する。これにより、$\mathtt{prog}_{\mathrm{obf}}$ と $\mathcal{C}$ の間で Diffie-Hellman 鍵 sk $(= g^{ab})$ が共有できる。$\mathcal{C}$ は、鍵 sk を用いて履歴依存プログラム f の暗号文 $ct = \mathrm{AE.Enc}_{sk}(f)$ を作成する。なお、$\mathrm{AE.Enc}_{sk}(f)$ は、秘密鍵 sk による f の暗号化関数である。

[46]セッション識別子は、プラットフォームが複数の同一プログラムをインストールする場合に、それらを互いに区別するために用いる。

[47]$\mathtt{prog}_{\mathrm{obf}}$ 内部で記憶（store）された sk は、$\mathcal{G}_{\mathrm{att}}$ の変数 $\mathtt{mem}$ の更新として出力され、記憶領域 T に保存される。

[48](g^a, g^b) の一方の g^a は $\mathcal{C}$ の作成した公開鍵、他方の g^b は $\mathtt{prog}_{\mathrm{obf}}$ 側で作成した公開鍵である。ステップ④では $\mathtt{prog}_{\mathrm{obf}}$ が Diffie Helman 鍵 $sk = (g^a)^b$ を計算したが、ステップ⑥でも $\mathcal{C}$ が持つ a と g^b を併せて、同じ Diffie Helman 鍵 $sk = (g^b)^a$ を計算し、両者の共有鍵とする。

⑦ 暗号化した関数を送信：$\mathcal{C}$ は、$\mathcal{S}$ に、(“obfuscate”, ct) を送信する。

⑧ $\mathtt{prog}_{\mathrm{obf}}$ において関数を登録：$\mathcal{S}$ が $\mathtt{resume}(eid, (\text{“obfuscate”}, ct))$ の実行を $\mathcal{G}_{\mathrm{att}}$ に指示すると、$\mathtt{prog}_{\mathrm{obf}}$ にメッセージ (“obfuscate”, ct) が送信される。$\mathtt{prog}_{\mathrm{obf}}$ 内部では、鍵 sk を用いて ct から f が求められ、$(f, \mathtt{st} := \bot)$ が格納される。ここで $\mathtt{st}$ は履歴依存プログラム f の内部記憶であり、この時点でメモリ領域が確保され、初期値として未定義値 $\bot$ を想定している。問題がなければ OK を出力する。$\mathtt{st}$ の値は $\mathtt{mem}$ に代入され、$\mathcal{G}_{\mathrm{att}}$ の記憶領域 T に保存される。f の出力である OK には署名 σ が付され、計算に用いた $\mathtt{prog}_{\mathrm{obf}}$ や sid, eid と共に以下の形式で出力される。また、この出力は $\mathcal{C}$ に送信される。

$$(sid, eid, \mathtt{prog}_{\mathrm{obf}}, \mathrm{OK}, \sigma) \tag{9}$$

（ロ）難読化した履歴依存プログラム f の実行

① $\mathcal{C}$ から $\mathcal{S}$ に「x を入力として計算せよ」とのメッセージ (“compute”, x) が届くと、$\mathcal{G}_{\mathrm{att}}$ の $\mathtt{resume}(eid, (\text{“compute”}, x, \bot))$ を起動し、eid で指定されたエンクレーブを使って $f(x, \bot)$ の計算[49]を要求する。

② $\mathcal{G}_{\mathrm{att}}$ は eid に対応する $\mathtt{prog}_{\mathrm{obf}}$ を T から取り出し、$\mathtt{prog}_{\mathrm{obf}}$ に $(\text{“compute”}, x, \bot))$ メッセージを送る。$\mathtt{prog}_{\mathrm{obf}}$ は以下の式を実行し、出力 y と更新後の状態 $\mathtt{st}'$ を得る。

$$(y, \mathtt{st}') = f(x, \mathtt{st})$$

$\mathtt{prog}_{\mathrm{obf}}$ の出力は実行証跡 σ と共に以下の形式で表され、$\mathcal{G}_{\mathrm{att}}$ の $\mathtt{outp}$ に反映される。

$$(sid, eid, \mathtt{prog}_{\mathrm{obf}}, y, \sigma) \tag{10}$$

また、更新後の状態 $\mathtt{st}'$ は $\mathcal{G}_{\mathrm{att}}$ の $\mathtt{mem}$ に出力され、記憶領域 T に保存される。

以上から、実行証跡付セキュアプロセッサの機能として $\mathcal{G}_{\mathrm{att}}$ が実現されていれば、以下が達成できることが示される。

1. $\mathcal{G}_{\mathrm{att}}$ に導入したプログラム（例えば $\mathtt{prog}_{\mathrm{obf}}$）との間に、Diffie-Hellman 鍵共有等によりセキュアチャネルを構築できる[50]。

2. プログラムはオープンソースプログラムでも良い（$\mathtt{prog}_{\mathrm{obf}}$ はオープンソースの例）。

   前項 1 の性質より、オープンソースで配布したプログラムであっても、後から実行証跡を確認することで、セキュアチャネルを構築できる。そのため、デジタル通貨ウォレットやマイナンバーカードによる個人認証のためのプログラムをオープンソースで配布しておけば、利用者は手元にある実行証跡付セキュアプロセッサを用いてプログラムを導入することができる。オープンソースであれば、プログラム機能が多くの人の目に触れることから、実行証跡付セキュアプロセッサ内部で実行されているプログラムの動作への信頼を確保しやすいというメリットもある。

[49] ここで、$\bot$ は証明で用いるために技巧的に導入された入力で、$y \neq \bot$ を入力すると $y = f(x, y)$ のように、第 2 入力の y がそのまま f の出力となる。ここでは詳細を割愛する。

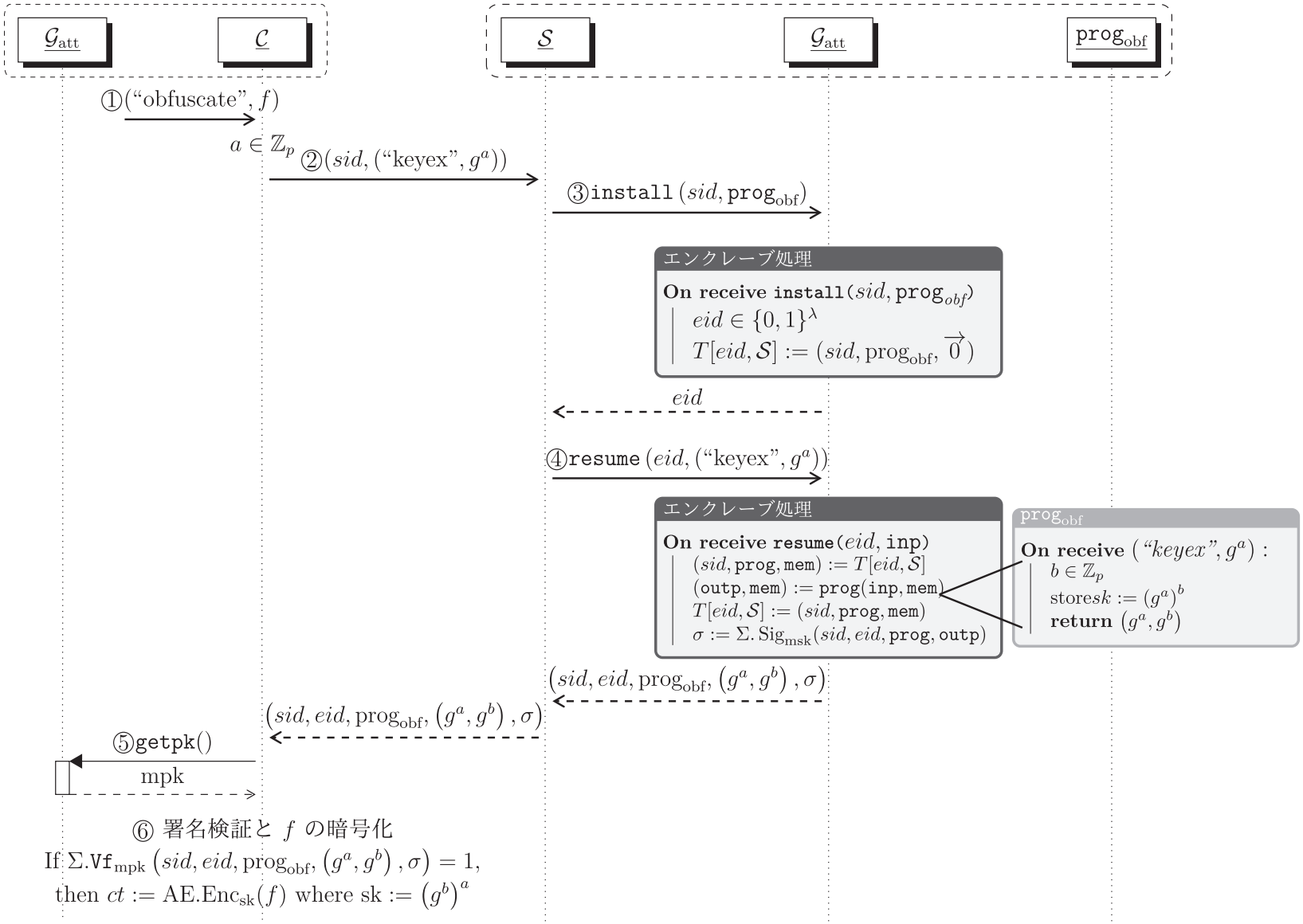
[50] 実行証跡付セキュアプロセッサでは、$\mathcal{G}_{\mathrm{att}}$ の実行証跡を確認することにより、$\mathtt{prog}_{\mathrm{obf}}$ の生成した鍵であることを外部から確認できる。実行証跡付セキュアプロセッサを用いない一般的な構成では、プログラムを IC カード等に組み込んで、IC カードの中に鍵を入れて配布するなどの手順を踏む必要がある。

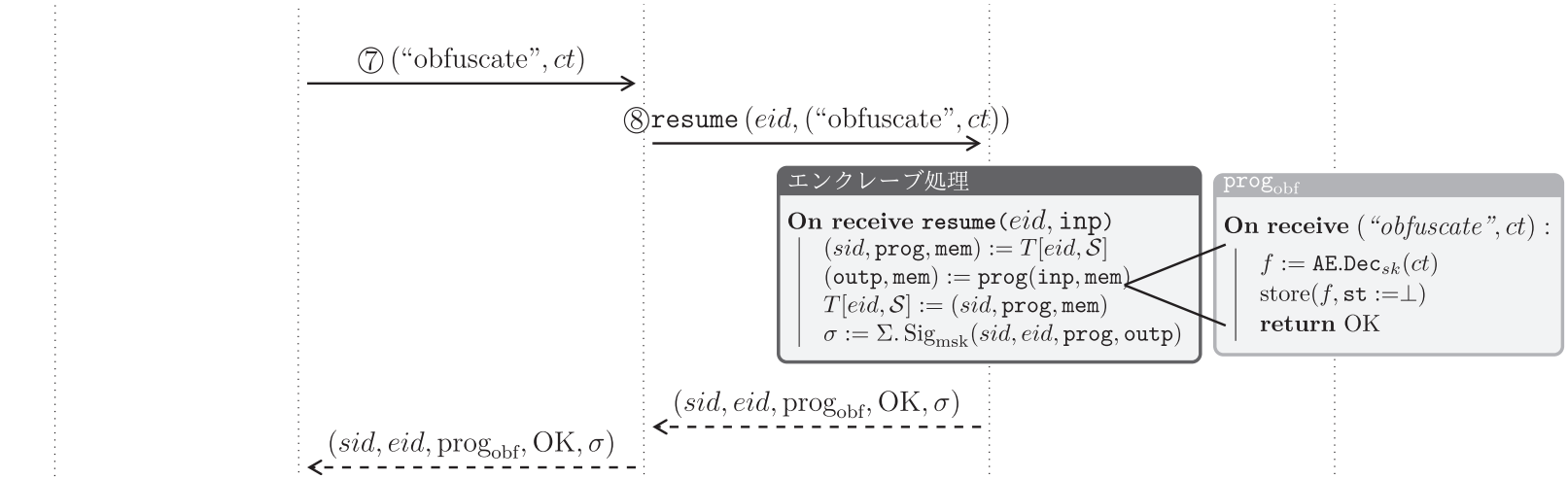
3. オープンソースプログラム $\mathtt{prog}_{\mathrm{obf}}$ を介して、任意の履歴依存プログラム f を難読化して $\mathcal{G}_{\mathrm{att}}$ に実行させることができる。

実行証跡付セキュアプロセッサをベースに画期的な機能や性能を実現する研究も現れている。例えば、Ekiden(Cheng et al., 2019) は、実行証跡付セキュアプロセッサを用いて、Ethereum のスマートコントラトの実行状態の秘匿を達成し、同時にスループットを数百倍に向上させている。これにより、資産残高や企業秘密を秘匿したスマートコントラクトを実現できる。Ekiden では、実行証跡付セキュアプロセッサの内部状態を暗号化してブロックチェーンに掲示することで、実行証跡付セキュアプロセッサ間での内部状態の共有を実現するといった興味深いアイディアが提案されている[51]。

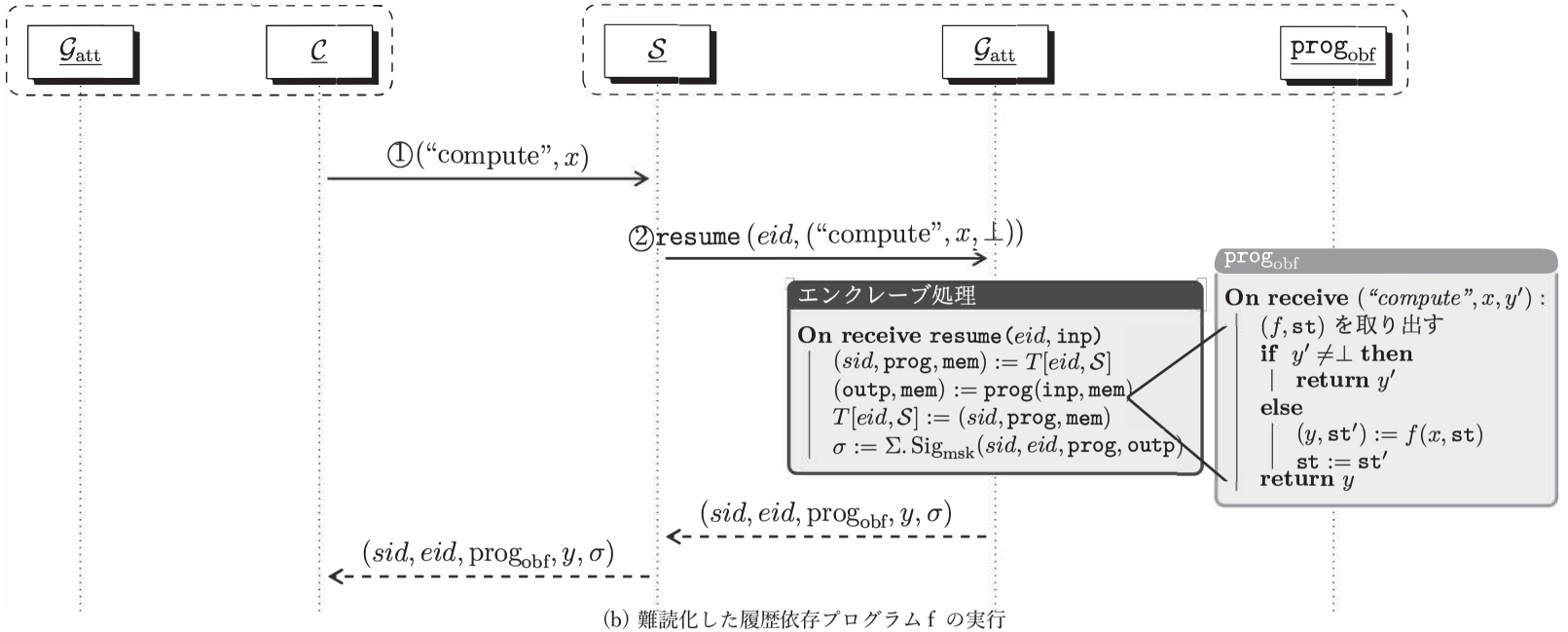
また、これまでの節で紹介したデジタル通貨ウォレットやオブザーバーは、$\mathcal{G}_{\mathrm{att}}$ のオープンソースプログラムとして実現できる。以下の節では、実行証跡付セキュアプロセッサの機能 $\mathcal{G}_{\mathrm{att}}$ を利用して匿名性と柔軟な透明性を実現する方式について述べる。

[51] 更新後の内部状態が暗号文されてブロックチェーンに掲載されていることを Proof of Publication と呼ばれる手法で実行証跡付セキュアプロセッサに検証させ、受理された時のみ、次の状態更新を許可することで、一貫性を保つ形で共有状態を実現している。


$\mathcal{G}_{\mathrm{att}}$
$\mathcal{C}$
$\mathcal{S}$
$\mathcal{G}_{\mathrm{att}}$
$\mathtt{prog}_{\mathrm{obf}}$
①(“obfuscate”, f)
$a \in \mathbb{Z}_p$ ②$(sid, (\text{“keyex”}, g^a))$
③$\mathtt{install}\,(sid, \mathtt{prog}_{\mathrm{obf}})$
エンクレーブ処理
On receive $\mathtt{install}(sid, \mathtt{prog}_{obf})$
$eid \in \{0,1\}^\lambda$
$T[eid, \mathcal{S}] := (sid, \mathrm{prog}_{\mathrm{obf}}, \overrightarrow{0})$
eid
④$\mathtt{resume}\,(eid, (\text{“keyex”}, g^a))$
エンクレーブ処理
On receive $\mathtt{resume}(eid, \mathtt{inp})$
$(sid, \mathtt{prog}, \mathtt{mem}) := T[eid, \mathcal{S}]$
$(\mathtt{outp}, \mathtt{mem}) := \mathtt{prog}(\mathtt{inp}, \mathtt{mem})$
$T[eid, \mathcal{S}] := (sid, \mathtt{prog}, \mathtt{mem})$
$\sigma := \Sigma.\mathrm{Sig}_{\mathrm{msk}}(sid, eid, \mathtt{prog}, \mathtt{outp})$
$\mathtt{prog}_{\mathrm{obf}}$
On receive $(\text{“keyex”}, g^a)$:
$b \in \mathbb{Z}_p$
store$sk := (g^a)^b$
return (g^a, g^b)
$\left(sid, eid, \mathrm{prog}_{\mathrm{obf}}, \left(g^a, g^b\right), \sigma\right)$
$\left(sid, eid, \mathrm{prog}_{\mathrm{obf}}, \left(g^a, g^b\right), \sigma\right)$
⑤$\mathtt{getpk}()$
mpk
⑥ 署名検証と f の暗号化
If $\Sigma.\mathtt{Vf}_{\mathrm{mpk}}\left(sid, eid, \mathrm{prog}_{\mathrm{obf}}, \left(g^a, g^b\right), \sigma\right) = 1$,
then $ct := \mathrm{AE.Enc}_{\mathrm{sk}}(f)$ where $\mathrm{sk} := \left(g^b\right)^a$



(a) 難読化した履歴依存プログラム f の登録



(b) 難読化した履歴依存プログラム f の実行

図 12: $\mathcal{G}_{\text{att}}$ を用いた内部記憶を持つプログラムの難読化の構成

(3) 実行証跡付セキュアプロセッサによる匿名性と透明性の両立

これまで見てきたように、実行証跡付セキュアプロセッサの理想機能として定義される4つのインターフェイス（初期化:initialize、署名検証鍵の取得:getpk、プログラムの登録:install、プログラムの実行:resume）を耐タンパーデバイスに実装するだけで、オープンソースプログラムとの鍵共有、履歴依存プログラムの仮想ブラックボックス難読化といった、非常に強力なセキュリティ機能を耐タンパーデバイスに持たせることができる。これらの機能を利用すれば、プログラムの形でさまざまな社会的ルールを柔軟に実装できるため、匿名性とAML/CFT等の透明性を両立するデジタル通貨の技術基盤を担うことができると考えられる。

その実現例として、実行証跡付セキュアプロセッサを用いてオブザーバー方式を拡張したデジタル通貨の構成例を図13に示し、その実行シーケンス例を図14に示す。この構成は以下の特徴を持つことを想定している。

1. ウォレット $\mathtt{prog}_{\mathcal{W}}$ 及びオブザーバー $\mathtt{prog}_{\mathcal{O}}$ はオープンソースで配布され、利用者や店舗が信頼するプログラムを選択して利用する。

2. $\mathtt{prog}_{\mathcal{O}}$、$\mathtt{prog}_{\mathcal{W}}$ は $\mathcal{G}_{\mathrm{att}}$ に install して実行証跡付セキュアプロセッサ内で実行される。

3. 実行結果には、プログラム名と共に実行証跡として匿名のデジタル署名（DAA、EPID等）が施される。ただし、二重支払等のルール違反が発覚した場合には、当該デバイスの発行するデジタル署名の匿名性が解除する鍵が公開され、当該デバイスから支払われたデジタル通貨はこの鍵を用いて違反デバイスを特定することにより銀行や店舗で受け取りを拒否できるとする。

4. $\mathtt{prog}_{\mathcal{O}}$ は二重支払いの防止等の安全性と透明性等の社会要請[52]を守ることを主目的とする。

5. $\mathtt{prog}_{\mathcal{W}}$ は利用者の資産とプライバシー[53]等の利用者の権利を守ることを主目的とする。

実行証跡付セキュアプロセッサの製造や供給は少数の企業や場合によっては政府機関に限られることが想定される。こうした製造者・供給者は、利用者の資産やプライバシーを操作する権限を持つ可能性があることから、実行される機能に対する信頼を確保することが極めて重要である。例の構成では、$\mathcal{G}_{\mathrm{att}}$ の機能を用いてウォレット $\mathtt{prog}_{\mathcal{W}}$ 及びオブザーバー $\mathtt{prog}_{\mathcal{O}}$ をオープンソースで公開可能とすることで、実行される機能の信頼の確保を試みている。

$\mathtt{prog}_{\mathcal{W}}$ は2節(2)で述べた鍵管理機能や、紛失／盗難時の対応、生体認証機能、WYSIWYS機能などを備えた基本的なデジタル通貨ウォレット機能を達成するプログラムを想定している。これに対して、$\mathtt{prog}_{\mathcal{O}}$ は監督当局が求める透明性を匿名支払条件として組み込んだプログラムであり、取引毎に匿名支払条件を満たすか否かの判定を実施し、条件を満たさない場合は当局への報告などの必要な義務を履行させることを想定している。すなわち、オブザーバー方式のように、二重支払いを未然に防ぐ機能に限定することなく、匿名支払条件を自由に $\mathtt{prog}_{\mathcal{O}}$ にプログラムし、匿名支払条件を満たす時に限り支払プロトコルに協力するように条件を拡張することを考える[54]。

[52]監督当局が、使用可能なオープンソースプログラムを複数認定し、そのリストを事前に公開していることを想定している。

[53]取引の相手や内容に関するプライバシーは、$\mathtt{prog}_{\mathcal{O}}$ に対しても秘匿することが望ましい。

[54]Brands らのトークン型デジタル通貨 (Brands, 1993) をベースに議論を進めてきたが、実行証跡付セキュ

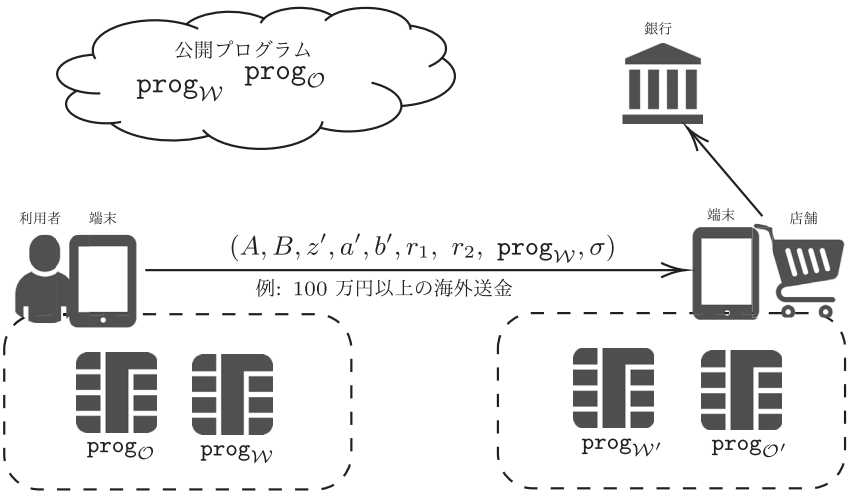


図 13: 実行証跡付セキュアプロセッサによる柔軟な匿名性と透明性のバランス（構成例）

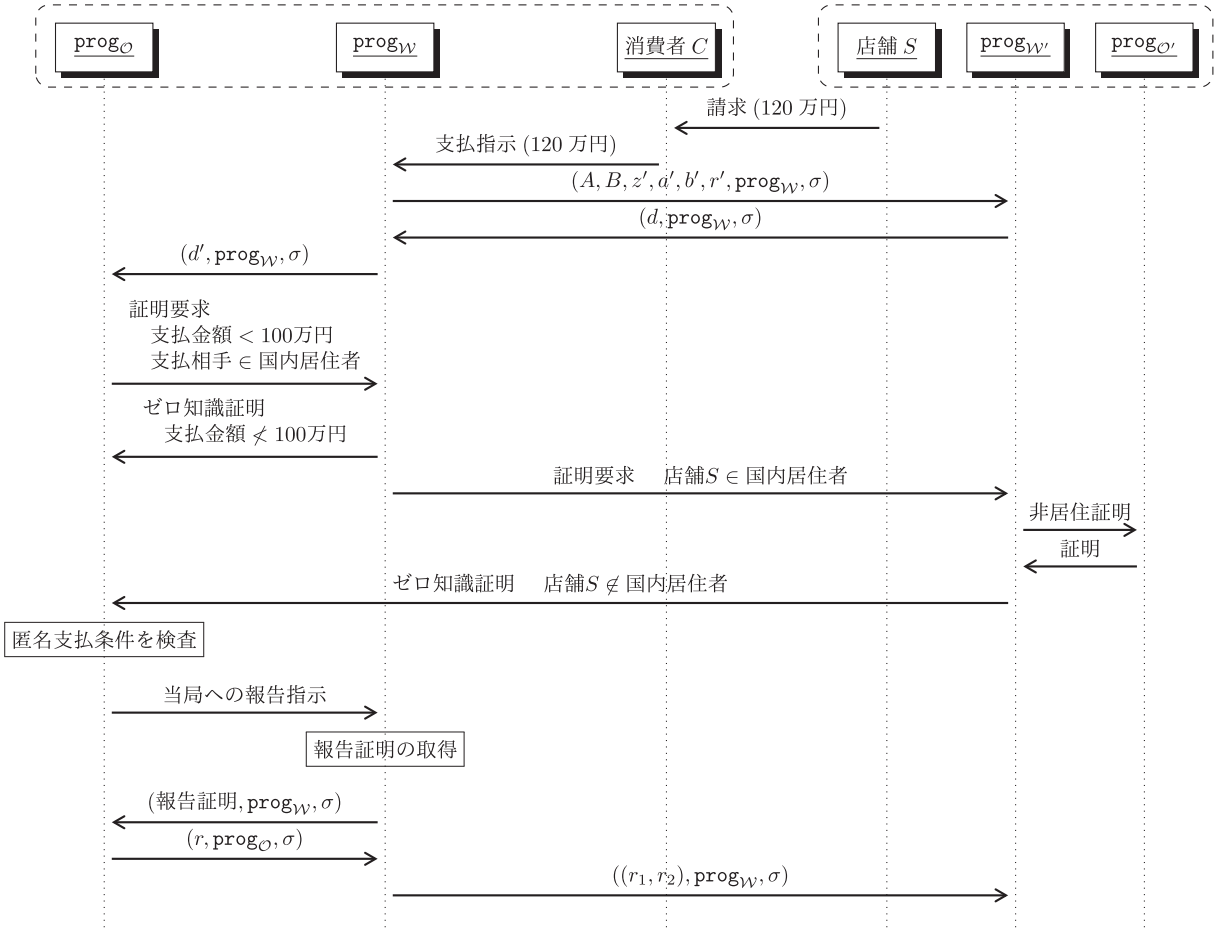


図 14: 実行証跡付セキュアプロセッサによる柔軟な匿名性と透明性のバランス（シーケンス例）

例として、100万円以上の国際送金では支払者IDの報告を必要とする条件を想定する。すなわち、100万円未満もしくは国内送金である取引は匿名で実施し、条件を満たさない場合は監督当局への報告を義務化するケースを考える[55]。ただし、$\mathtt{prog}_{\mathcal{O}}$が取引内容に直接アクセス可能な場合は匿名支払条件の判定は自明だが、プライバシー保護の観点から、$\mathtt{prog}_{\mathcal{O}}$に全ての取引内容を開示することは望ましくない。そこで下記プロトコルでは$\mathtt{prog}_{\mathcal{O}}$が「支払金額 $<$ 100万円」や「支払相手 $\in$ 国内居住者」などの匿名支払条件を満たすか否かの証明を利用者や店舗の$\mathtt{prog}_{\mathcal{W}}$に対して要求し、対応する$\mathtt{prog}_{\mathcal{W}}$が条件の成立／不成立のみをゼロ知識証明で伝える構成としている。同様に、店舗$\mathcal{S}$も$\mathtt{prog}_{\mathcal{W}'}$, $\mathtt{prog}_{\mathcal{O}'}$を$\mathcal{G}_{\mathrm{att}}$にインストールしており、$\mathtt{prog}_{\mathcal{W}'}$は$\mathcal{S}$の資産／プライバシーを保護することを目的にプログラムされており、$\mathtt{prog}_{\mathcal{O}'}$は$\mathcal{S}$がルールを遵守して虚偽の証明を行わないことを監視していると想定している。例では、国内居住者でないことを証明するにあたり、$\mathtt{prog}_{\mathcal{W}'}$による非居住証明に$\mathtt{prog}_{\mathcal{O}'}$が協力する様子を示している[56]。

匿名支払条件を満たす場合は、オブザーバー方式によるプロトコルに従って支払を実行する。匿名支払条件を満たさない場合は、$\mathtt{prog}_{\mathcal{W}}$に対して監督当局への報告を要求する。その際、$\mathtt{prog}_{\mathcal{W}}$は取引内容を監督当局に報告し、監督当局から報告済みであることの証明書（報告証明）を取得して$\mathtt{prog}_{\mathcal{O}}$に提示する。これを受けて$\mathtt{prog}_{\mathcal{O}}$がプロトコル実行に協力し、国際送金が実行される。具体的なプロトコルのシーケンス例は以下のとおりである（図14参照）。

- 店舗Sは、利用者Cに120万円を請求する。
- 利用者Cは、$\mathtt{prog}_{\mathcal{W}}$に120万円の支払いを指示する。
- $\mathtt{prog}_{\mathcal{W}}$は、追跡可能Schnorrブラインド署名により120万円のトークン(A, B, z', a', b', r')を生成し、トークン、$\mathtt{prog}_{\mathcal{W}}$、実行証跡$\sigma$を$\mathtt{prog}_{\mathcal{W}'}$に送信する。$\mathtt{prog}_{\mathcal{W}'}$は、乱数$d$、$\mathtt{prog}_{\mathcal{W}}$、実行証跡$\sigma$を$\mathtt{prog}_{\mathcal{W}}$に返信する。
- $\mathtt{prog}_{\mathcal{W}}$は、$\mathtt{prog}_{\mathcal{O}}$に乱数$d$'、$\mathtt{prog}_{\mathcal{W}}$、実行証跡$\sigma$を送信する。
- $\mathtt{prog}_{\mathcal{O}}$は$\mathtt{prog}_{\mathcal{W}}$に対して、支払金額が100万円未満であること、もしくは支払相手が国内居住者であることの条件を満たしているか否かの証明を要求する。
- $\mathtt{prog}_{\mathcal{W}}$は支払い金額が100万円未満でないことをゼロ知識証明で示す。
- $\mathtt{prog}_{\mathcal{W}}$は$\mathtt{prog}_{\mathcal{W}'}$に対し、支払相手となる店舗$S$が国内居住者であるか否かの証明を要求する。

アプロセッサに内蔵するプログラムは、様々なデジタル通貨方式に合わせて設計できるため、ブロックチェーン上で実装される分散台帳をベースとするデジタル通貨に対しても適用可能であると見られる。例えば、高橋らは、実行証跡付セキュアプロセッサに基づいて非中央集権型ブロックチェーン上で転々流通性を持つマイクロペイメント（100円以下の少額決済を実現する電子決済技術。ブロックチェーンでは取引をブロックチェーンに登録するために相対的に高額な手数料が必要となるため、複数の決済をまとめて登録する技術や確率的に少数の決済のみを選択して登録する技術などが提案されている）を実現している (Takahashi and Otsuka, 2021)。

[55]利用者と店舗がそれぞれ$\mathtt{prog}_{\mathcal{W}}$, $\mathtt{prog}_{\mathcal{O}}$を公開プログラムから選択して、それぞれの端末の$\mathcal{G}_{\mathrm{att}}$にインストールしている状況を想定する。利用者と店舗の$\mathtt{prog}_{\mathcal{W}}$,$\mathtt{prog}_{\mathcal{O}}$を区別するため、店舗がインストールしたプログラムを$\mathtt{prog}_{\mathcal{W}'}$,$\mathtt{prog}_{\mathcal{O}'}$と表記する。本来、これらは同一種類のプログラムである必要はなく、それぞれが任意に公開プログラムから選択してインストールすると仮定すべきであるが、その際には互いに相手のプログラムがルールに沿ったものであるか否かを検証する必要が生じ、複雑になるため本節では単純化して、同一種類の$\mathtt{prog}_{\mathcal{W}}$、$\mathtt{prog}_{\mathcal{O}}$がインストールされていると想定する。

[56]国内非居住者であることの証明には、住民票等の公的機関が発行した検証可能証明書（Verifiable Credential）が必要だが、検証可能証明書は多くのプライバシー情報を含むため、本人を特定できない形で非居住者であるという事実だけを証明する属性証明書あるいは派生証明書（Derived Credential）だけを提示することが望ましい。このためには、派生証明書が検証可能証明書から正しく作られたことを保証する仕組みが必要であり、その仕組みについては自己主権型アイディンティティ（Self-Sovereign Identity）等の概念で盛んに研究されている。実行証跡付セキュアプロセッサに基づく構成法については (Moriyama and Otsuka, 2022a,b) 等の研究がある。

- $\mathtt{prog}_{\mathcal{W}'}$ は $\mathtt{prog}_{\mathcal{O}}$ に店舗 S が国内居住者でないことをゼロ知識証明で示す。
- $\mathtt{prog}_{\mathcal{O}}$ は匿名支払条件を検査する。
- $\mathtt{prog}_{\mathcal{O}}$ は $\mathtt{prog}_{\mathcal{W}}$ に当局への報告を指示する。
- $\mathtt{prog}_{\mathcal{W}}$ は報告証明の取得を行う。
- $\mathtt{prog}_{\mathcal{W}}$ は $\mathtt{prog}_{\mathcal{O}}$ に報告証明、$\mathtt{prog}_{\mathcal{W}}$、実行証跡 σ を送付する。
- $\mathtt{prog}_{\mathcal{O}}$ は $\mathtt{prog}_{\mathcal{W}}$ に乱数と実行証跡を送付する。
- $\mathtt{prog}_{\mathcal{W}}$ は $\mathtt{prog}_{\mathcal{W}'}$ に 2 つの乱数 (r_1, r_2)、$\mathtt{prog}_{\mathcal{W}}$、実行証跡 σ を送付する。

5 まとめ

本稿では、デジタル通貨の安全性のほか、匿名性と透明性の両立についてウォレット技術の最新動向から概観した。具体的には、まず、スマートフォンに搭載されている耐タンパーデバイスを活用して、デジタル通貨の保管や移動に必要となるデータの保護と、取引実行時における本人意思の確認を行う方法について整理を行った。IC カードやコールドウォレットは、利用時のみ PC 等と接続して利用することから、ネットワークに接続していない限りサイバー攻撃を受けるリスクはないが、スマートフォンや PC 端末に内蔵された耐タンパーデバイスは常時インターネットに接続されている状態であり、耐タンパー性を維持するための対策が求められる。近年は、プログラム可能な耐タンパーデバイスがスマートフォンや PC 端末等を中心に広く普及しており、デジタル通貨のウォレットとしての活用が大いに期待されるところである。

さらに、こうした耐タンパーデバイスを用いて二重支払いを未然に防止するオブザーバー方式を紹介したうえで、近年提案された実行証跡付セキュアプロセッサを用いたデジタル通貨スキームを提案した。オブザーバー方式は、二重支払いが行われたときにのみ利用者を特定可能にするという、固定されたアルゴリズムを前提とした方式であった。デジタル通貨スキームに対する AML/CFT 等の社会要請を踏まえれば、より柔軟な透明性を実現できることが望ましい。もちろん、社会要請を満たす透明性条件を明確に定義できれば、今後の研究の進展によって、匿名性と透明性を両立する技術について、社会的に合意の取れた技術が出現する可能性は残されている。

本稿で紹介した実行証跡付セキュアプロセッサは、耐タンパー技術に関する強い仮定を必要とするものの、導入するプログラムの特徴を柔軟に設計することで多様な AMT/CFT 対応ルールをオブザーバーに設定できることから、各種の匿名デジタル通貨に透明性を導入でき、社会的に合意のとれた匿名デジタル通貨システムの構築に道を拓くものである。本稿の議論が、デジタル通貨の発展の一助となれば幸甚である。

参考文献

Abe, Masayuki and Miyako Ohkubo (2001) "Provably Secure Fair Blind Signatures with Tight Revocation," in *Advances in Cryptology - ASIACRYPT 2001*, Vol. 2248 of Lecture Notes in Computer Science, pp. 583–602.

Baldimtsi, Foteini and Anna Lysyanskaya (2013) "On the Security of One-Witness Blind Signature Schemes," *Advances in Cryptology - ASIACRYPT 2013*, Vol. 8270, pp. 82–99.

Barak, Boaz, Oded Goldreich, Russell Impagliazzo, Steven Rudich, Amit Sahai, Salil P. Vadhan, and Ke Yang (2001) "On the (Im)possibility of Obfuscating Programs," in *Advances in Cryptology - CRYPTO 2001*, Vol. 2139 of Lecture Notes in Computer Science, pp. 1–18.

Ben-Sasson, Eli, Alessandro Chiesa, Christina Garman, Matthew Green, Ian Miers, Eran Tromer, and Madars Virza (2014) "Zerocash: Decentralized Anonymous Payments from Bitcoin (Extended)," *2014 IEEE Symposium on Security and Privacy*, pp. 459–474.

Brands, Stefan (1993) "Untraceable Off-line Cash in Wallets with Observers (Extended Abstract)," in *Advances in Cryptology - CRYPTO '93*, Vol. 773 of Lecture Notes in Computer Science, pp. 302–318.

Brickell, Ernie, Jan Camenisch, and Liqun Chen (2004) "Direct Anonymous Attestation," *Proceedings of the 11th ACM Conference on Computer and Communications Security*, pp. 132–145.

Brickell, Ernie and Jiangtao Li (2010) "Enhanced Privacy ID from Bilinear Pairing for Hardware Authentication and Attestation," *2010 IEEE Second International Conference on Social Computing*, pp. 768–775.

Bünz, Benedikt, Shashank Agrawal, Mahdi Zamani, and Dan Boneh (2020) "Zether: Towards Privacy in a Smart Contract World," in *Financial Cryptography and Data Security*, pp. 423–443.

Chaum, David (1982) "Blind Signatures for Untraceable Payments," in *Advances in Cryptology: Proceedings of CRYPTO '82*, pp. 199–203.

Chaum, David, Christian Grothoff, and Thomas Moser (2021) "How to Issue a Central Bank Digital Currency," *ArXiv*, Vol. abs/2103.00254.

Chaum, David and Torben Pryds Pedersen (1993) "Wallet Databases with Observers," in *Advances in Cryptology — CRYPTO' 92*, pp. 89–105.

Chaum, David, Bert den Boer, Eugène van Heyst, Stig Mjølsnes, and Adri Steenbeek (1990) "Efficient Offline Electronic Checks," in *Advances in Cryptology — EUROCRYPT '89*, pp. 294–301.

Cheng, Raymond, Fan Zhang, Jernej Kos, Warren He, Nicholas Hynes, Noah Johnson, Ari Juels, Andrew Miller, and Dawn Song (2019) "Ekiden: A Platform for Confidentiality-Preserving, Trustworthy, and Performant Smart Contract Execution," *2019 IEEE European Symposium on Security and Privacy (EuroS&P)*, pp. 185–200.

Feige, Uriel, Amos Fiat, and Adi Shamir (1988) "Zero-knowledge proofs of identity," *Journal of Cryptology*, Vol. 1, No. 2, pp. 77–94.

Fujisaki, Eiichiro and Tatsuaki Okamoto (1998) "Practical Escrow Cash Schemes," *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, Vol. E81-A, pp. 11–19.

GlobalPlatform (2014) "Secure Element Access Control v1.1," *GlobalPlatform Device Technology, Document Reference: GPD_SPE_013*.

Goldwasser, Shafi and Guy N. Rothblum (2007) "On Best-Possible Obfuscation," in *Theory of Cryptography*, Vol. 4392 of Lecture Notes in Computer Science, pp. 194–213.

GSM Association (2011) *GSMA NFC Handset APIs Requirement Specification Version 2.0*, pp. 1–48.

Heilman, Ethan, Leen AlShenibr, Foteini Baldimtsi, Alessandra Scafuro, and Sharon Goldberg (2017) "TumbleBit: An Untrusted Bitcoin-Compatible Anonymous Payment Hub," in *Network and Distributed System Security Symposium*: Internet Society.

Maxwell, Gregory (2015) "Confidential transactions," URL: `https://people.xiph.org/agreg/confidential_values.txt`.

Mitani, Tatsuo and Akira Otsuka (2020) "Traceability in Permissioned Blockchain," *IEEE Access*, Vol. 8, pp. 21573–21588.

Moriyama, Koichi and Akira Otsuka (2022a) "Attested Execution Secure Processor-based Architecture for Self-Sovereign Identity Systems Preserving Privacy," 暗号と情報セキュリティシンポジウム *(SCIS 2022)*.

Moriyama, Koichi and Akira Otsuka (2022b) "Permissionless Blockchain-Based Sybil-Resistant Self-Sovereign Identity Utilizing Attested Execution Secure Processors," *2022 IEEE International Conference on Blockchain (Blockchain '22)*.

National Computer Security Center, U.S. (1985) *Trusted Computer System Evaluation Criteria*.

Okamoto, Tatsuaki (1992) "Provably Secure and Practical Identification Schemes and Corresponding Signature Schemes," *Advances in Cryptology - CRYPTO '92*, Vol. 740, pp. 31–53.

Okamoto, Tatsuaki and Kazuo Ohta (1992) "Universal Electronic Cash," in *Advances in Cryptology — CRYPTO '91*, pp. 324–337.

Pass, Rafael, Elaine Shi, and Florian Tramér (2017) "Formal Abstractions for Attested Execution Secure Processors," in *Advances in Cryptology — EUROCRYPT 2017*, pp. 260–289.

Pointcheval, David and Jacques Stern (2000) "Security Arguments for Digital Signatures and Blind Signatures," *Journal of Cryptology*, Vol. 13, No. 3, pp. 361–396.

Seurin, Yannick (2012) “On the Exact Security of Schnorr-Type Signatures in the Random Oracle Model,” *EUROCRYPT*, Vol. 7237, pp. 554–571.

Stadler, Markus, Jean-Marc Piveteau, and Jan Camenisch (1995) “Fair Blind Signatures,” in *Advances in Cryptology - EUROCRYPT ’95*, Vol. 921 of Lecture Notes in Computer Science, pp. 209–219.

Takahashi, Taisei and Akira Otsuka (2021) “Probabilistic Micropayments with Transferability,” in *26th European Symposium on Research in Computer Security — ESORICS 2021*, Vol. 12972 of Lecture Notes in Computer Science, pp. 390–406.

Weigold, Thomas, Thorsten Kramp, Reto Hermann, Frank Höring, Peter Buhler, and Michael Baentsch (2008) “The Zurich Trusted Information Channel — An Efficient Defence Against Man-in-the-Middle and Malicious Software Attacks,” in *Trusted Computing and Trust in Information Technologies — Trust ’08*, pp. 75–91.

大塚玲 (2022) “ブロックチェーンを利用した暗号資産の安全性と匿名性：原理と限界,” 金融研究, Vol. 41, No. 1, pp. 1–56.

中山靖司 (1998) “電子マネー技術と特許,” 金融研究所ディスカッション・ペーパー, No. 1998-J-33.

中山靖司・太田和夫・松本勉 (1999) “電子マネーを構成する情報セキュリティ技術と安全性評価,” 金融研究, Vol. 18, No. 2, pp. 57–114.

中山靖司・森畠秀実・阿部正幸・藤崎英一郎 (1997) “電子マネーの一実現方式について,” 金融研究, Vol. 16, No. 2, pp. 75–86.